NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Canada

Performance Analysis and Optimized Design of CMOS Buffers

Yi Zhu

A Thesis

In

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering at
Concordia University
Montréal, Québec, Canada

July 1988

ISBN 0-315-44867-9

ABSTRACT

Performance Analysis and Optimized Design of CMOS Buffers

Yi Zhu

This thesis describes the theory and implementation of a module generator for CMOS buffers. The generator is written in C language and outputs optimal buffer designs in respect to an objective function and layout. The user has the choice of minimizing speed, power and area or a combination of these. Also the options to choose buffer configurations is provided, making this tool useful for various designs.

The module generator is made up of two parts the optimizer and the layout generator. The former is the essence of this thesis, where variations of process and design parameters with respect to each objective function is studied in detail and in each case optimal design parameters are derived.

ACKNOWLEDGEMENTS

I wish to thank my supervisor Dr. A. J. Al-Khalili for his guidance, encouragement and support which have been provided during the course of my research. His judicious suggestions made the successful completing of this thesis possible.

I wish to express my thanks to my girl friend Holly Chen for her help in typing.

I would finally like to thank Concordia VLSI lab for providing the facilities and pleasant research environment.

Dedicated to

my beloved parents

and Holly

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TBLES

LIST OF SYMBOLS

a :	Slope of step response
A_B :	Buffer area
A_d :	Diffusion area
A_g :	Gate area
A_I :	Area of a single CMOS inverter
$A_{N,P}$:	Process constant
$AD_{n,p}$:	Unit area of drain diffusion region
AT :	Area-time product
AT^2 :	Area and square of time product
b :	Slope of Ramp response
$B_{N,P}$:	Input waveform constant
C_c :	Interconnection capacitance
C_{db} :	Drain and bulk capacitance
C_{eqdb} :	Unit bottom equivalent junction capacitance
C_{eqds} :	Unit sidewall equivalent junction capacitance
C_{gb} :	Gate and bulk capacitance
C_{gd} :	Gate and drain capacitance
C_{gs} :	Gate and source capacitance
C_{gi} :	Gate capacitance
C_{in} :	Input capacitance
C_{load} :	Load capacitance
C_L :	Interstage load capacitance
C_{Nout} :	Output capacitance of last inverter
C_o :	Input capacitance of buffer
C_{out} :	Output capacitance

C_{ox}:	Unit gate capacitance
C_x:	Unit layout equivalent capacitor
C_T:	Interstage capacitance
$CJ_{n,p}$:	Unit area of zero-bias bulk junction bottom capacitance
$CJSW_{n,p}$:	Unit length of zero-bias bulk junction sidewall capacitance
f:	Frequency
g_i:	β dependent process constant
GND:	Ground
I:	Leakage current
$I_{n,p}$:	Transistor current
$i_{n,p}$:	Normalized transistor current
IS:	Bulk junction saturation current
k:	Boltzmams's constant
K_{eq}:	Dimensionless constant
$K_{N,P}$:	Transconductance parameter
L_c:	Length of interconnection layer
L_d:	Drain length
L_{ni}:	Length of N-channel transistor
$LD_{n,p}$:	Lateral diffusion
M:	Bulk junction bottom and sidewall grading coefficient
m:	Fan-out from the preceding stage
n:	Normalized threshold voltage of N-channel transistor
n:	Fan-out
N:	Number of inverter stages
N_o:	Optimum number of inverter stages
p:	Normalized threshold voltage of P-channel transistor
P:	Power dissipation

P_B: Buffer power dissipation

P_d: Dynamic dissipation

P_{sc}: Short-circuit power dissipation

P_t: Transient power dissipation

PD: Power-Delay product

$PD_{n,p}$: unit length of drain diffusion region

q: Electronic charge

S: Scaling factor

S_o: Optimum scaling factor

s_f: Slope for the fall time

s_r: Slope for the rise time

T: Temperature

t: Time

t_{di}: Propagation delay

t_{din}: Input load dependent delay

t_{do}: Fan-out dependent delay

t_{dr}: Ramp response delay

t_{ds}: Step response delay

t_{ox}: Oxide thickness

t_{Total}: Buffer propagation delay

V_{DD}: Working voltage

V_{in}: Input voltage

v_{in}: Normalized input voltage

v_{nl}: Input voltage when the N-transistor is entering the linear region

V_{out}: Output voltage

v_o: Normalized output voltage

V: Voltage across the diode

V_T: Threshold voltage

W_c: Width of interconnection layer

W_d: Drain width

W_{ni}: Width of N-channel transistor

Y: Buffer fan-out

β: Channel width ration between the P- and N-channel transistor

β_o: Optimum β

δ_i: β independent process constant

ϵ_{ox}: Gate permittivity

Φ_o: Bulk junction potential

γ_i: β independent process constant

λ: Design parameter constant

$\mu_{n,p}$: Surface mobility

τ_f: Fall time

τ_o: Unit delay

τ_r: Rise time

CHAPTER 1

INTRODUCTION

Many silicon compilers use the standard technique of place and route of cells. These cells are previously designed and characterized. However during the design a variety of cells are required particularly if the design is to be optimum. Many vendors provide numerous standard cells for the same function, however it is difficult to cater for the variety of cells that is required. This leads to too much storage and pre-compiler work and also due to limited choice the optimization will be poor. It is much more convenient to have an adaptive module generator that generates cells automatically to suit a required situation in terms of performance and layout. This thesis focuses on this type of generators.

The module generator optimizes its objective function and then produces its layout to that effect. The optimization function can be speed, power, area or a combination of these. Each of these three basic quantities can be traded off against each other: speed may be increased by consuming more power, power consumption can be decreased by reducing buffer size, etc. Much of the art of the design process is in balancing the tradeoffs: How to achieve one quantity at upper bound and other two at lower bounds ?

The module implements four major parts:

1. Performance optimization
   - It evaluates capacitances of a proposed buffer, to determine expected time delay and calculates power and area for the buffer under consideration.
   - It determines the effect of the input waveform, channel width ratio and the output loading.

- It calculates the performance parameters such as scaling factor or the number of inverter stages for a particular buffer.

2. Design parameters calculation
   - To determine the width of each inverter to form a buffer according to user's specification as obtained from part 1.
3. Poly routing configuration selection
   - To provide multiple configuration of layouts. The zigzag, parallel and star cross are available to be selected.
4. Layout generation
   - To provide a two-dimensional full custom mask geometry file in CIF format for CMOS-1B 5 micron *NT* process.

This module computes optimum buffer size to meet the performance requirements specified by the user. It assists the user evaluate rapidly many design alternatives choosing the best tradeoff of speed, power and area. It is a powerful and fast tool that allows the user to explore the space of designs having optimal buffer sizes. The rest of this thesis is organized as follows, *Chapter 2* exploring CMOS modelings which describes CMOS capacitances, CMOS propagation delay and delay analysis expressions for CMOS buffer. *Chapter 3* focuses on speed optimization of CMOS devices and describes the optimum ratio of channel width β for minimum delay for CMOS buffer. *Chapter 4* describes power consumption of CMOS device and overviews CMOS static, short-circuit and transient power consumption. In *Chapter 5*, we look into the optimization of CMOS device in terms of area, Power-Delay tradeoff, Area-Time and AT^2. *Chapter 6* describes the implementation of our module which is written in C followed by the conclusion and future work in *Chapter 7*.

CHAPTER 2

DEVELOPMENT OF A MODEL FOR THE CMOS INVERTER PROPAGATION DELAY

2.1 Introduction

As we know, today's VLSI silicon compiler must have accurate and efficient models of various cells used in the compiler to achieve a high degree of performance. In this chapter, two major models are presented for our analysis: a) CMOS inverter capacitance model and b) inverter timing delay model. Some MOS capacitance's models have been developed for MOS transistor [1] - [5]. Using these models, a general linear capacitance's model for a CMOS inverter is derived in terms of SPICE parameters and newly defined constants. This is described in Section 2.2. The capacitance's expression for a single CMOS inverter model provides the necessary and sufficient condition (e.g. the dynamic response of MOS circuit is very much dependent on the capacitance's model) to derive a precise timing model for CMOS inverter and buffer. Timing models for MOS transistor logic are available in the literatures [5] - [10]. Logic timing model for a ramp input waveform has been developed by N. Hedenstierna [11]. In Section 2.3 we develop the timing model of CMOS inverter for the ramp input waveform using capacitance's model described in Section 2.2. The timing model is further extented to cover multi-stage buffer design.

2.2 Capacitance for CMOS Inverter

The propagation delay of a MOS device is heavily dependent on it's parasitic and peripheral capacitance. Therefore, accurate capacitance's model provides the necessary and sufficient condition to estimate the transient delay of circuits.

First of all, we consider all parasitic capacitances of a MOS inverter and the interconnect capacitances which relate to the other MOS devices are ignored. Secondly, we assume that all capacitances in the MOS device are linerized to constant values. The model that is to be derived is an enhanced version of parameterized model.

2.2.1 Input Capacitance

In this section we study an inverter's input capacitances by considering the components of the input capacitance. The model of MOS transistor's parasitic capacitances is represented by a cross-section diagram in Fig. 2.2.1. For a MOS transistor, the input capacitance C_{in} is equivalent to the gate capacitance C_g.

$$C_{in} = C_g = C_{gb} + C_{gs} + C_{gd} \tag{2.2.1}$$

Where C_{gb} is the capacitance between gate and bulk or substrate

C_{gs} is the capacitance between gate and source

C_{gd} is the capacitance between gate and drain.

Therefore the input capacitance C_{in} for a CMOS inverter is lumped by the sum of NMOS and PMOS's gate capacitances according to Fig. 2.2.2 which shows the CMOS inverter's model with parasitic capacitances. Fig. 2.2.3 shows a top-view of a CMOS inverter channel layout. Using Fig. 2.2.1 and Fig. 2.2.3 C_{in} is then related to devices size and SPICE parameters.

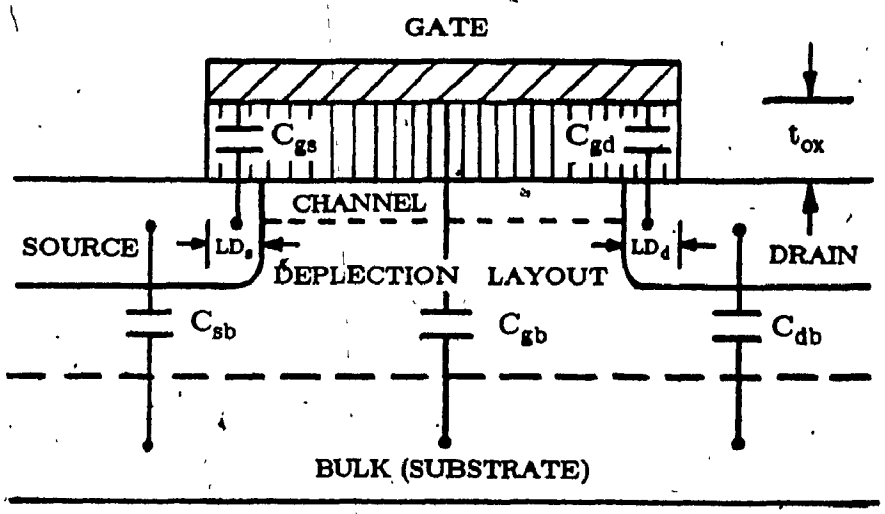


Figure 2.2.1 Parasitic Capacitances of a MOS Transistor

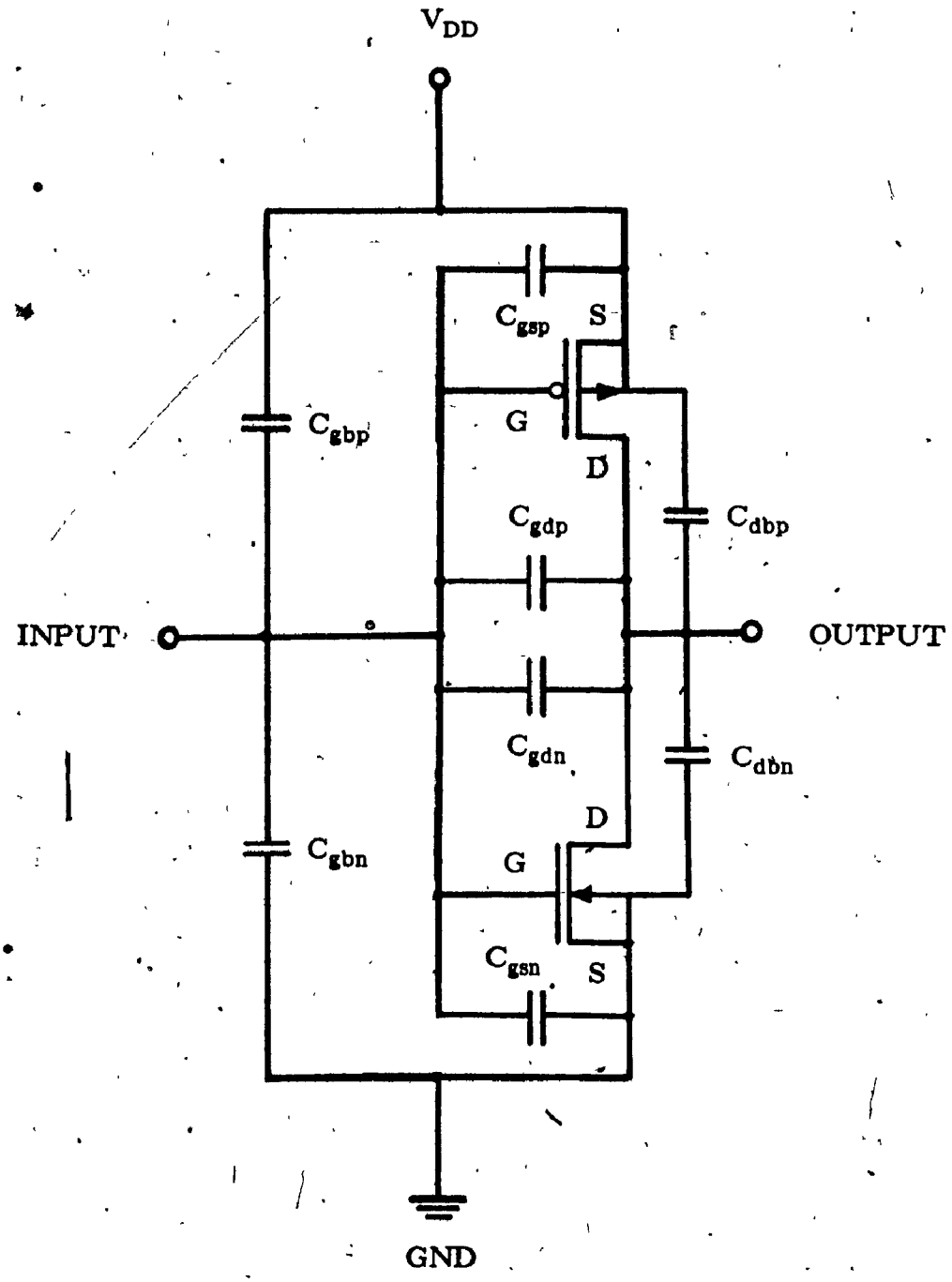


Figure 2.2.2 Capacitance Symbols of a CMOS Inverter's Model

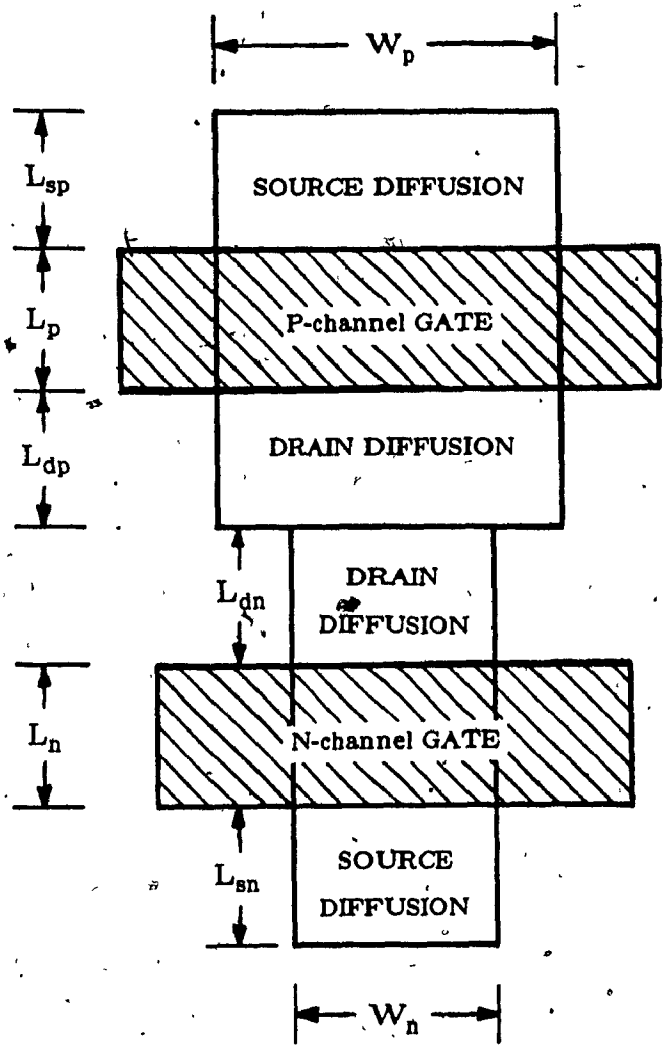


Figure 2.2.3 Top-view of a CMOS Inverter Channel Layout

$$C_{in} = C_{gbn} + C_{gbp} + C_{gsn} + C_{gsp} + C_{gdn} + C_{gdp}$$

$$= C_{ozn} W_n L_n + C_{ozp} \beta W_n L_p$$

$$+ C_{ozn} W_n LD_{sn} + C_{ozp} \beta W_n LD_{sp}$$

$$+ C_{ozn} W_n LD_{dn} + C_{ozp} \beta W_n LD_{dp} \qquad (2.2.2)$$

Where $\beta = W_p / W_n$ is the channel width ratio between the P- and N-channel transistors.

LD_{sn} is a n-type lateral diffusion of gate-source overlap

LD_{sp} is a P-type lateral diffusion of date-source overlap

LD_{dn} is a n-type lateral diffusion of gate-drain overlap

LD_{dp} is a P-type lateral diffusion of gate-drain overlap

C_{ozn} and C_{ozp} are the gate capacitances per unit area for both of N- and P-channel transistors and are defined in appendix B.

Usually both transistors have the same gate oxide thickness $C_{ox} = C_{ozn} = C_{ozp}$, the same channel length $L = L_n = L_p$ and the same type of lateral diffusion, i.e. $LD_n = LD_{sn} = LD_{dn}$, $LD_p = LD_{sp} = LD_{dp}$ (refer to Northern Telecom SPICE process parameters for example). The input capacitance equation (2.2.2) can now be represented in terms of device geometry constants

$$C_{in} = C_{gn} (\delta_1 + \delta_2 \beta) \qquad (2.2.3)$$

where $C_{gn} = C_{ox} W_n L_n$ is the gate capacitance of the N-channel transistor and W_n and L_n are the width and length of N-channel transistor respectively as shown in Fig. 2.2.3. δ_1 and δ_2 both depend on process, design rule and defined here as

$$\delta_1 = 1 + 2\frac{LD_n}{L} \qquad (2.2.4)$$

and

$$\delta_2 = 1 + 2\frac{LD_p}{L} \qquad (2.2.5)$$

and can be evaluated for a particular process. The typical values of δ_1 and δ_2 for given process are in the Table A-2 of Appendix A.

2.2.2 Output Capacitance

The output capacitance of CMOS inverter also affect the speed of CMOS device. The output capacitance C_{out} may be represented by the sum of the drain-bulk capacitances C_{dbp} and C_{dbn} of both N- and P-channel transistors. Using the capacitance model of the inverter given in Fig 2.2.2 linearized C_{out} may be represented as

$$\begin{aligned} C_{out} &= C_{dbn} + C_{dbp} \\ &= (C_{eqdbn} AD_n + C_{eqdsn} PD_n) \\ &\quad + (C_{eqdbp} AD_p + C_{eqdsp} PD_p) \end{aligned} \qquad (2.2.6)$$

Where C_{eqdbn} and C_{eqdbp} are the bottom equivalent junction capacitances of drain diffusion per unit area for both channel transistors.

C_{eqdsn} and C_{eqdsp} are the sidewall equivalent junction capacitances of drain diffusion region per unit length of junction perimeter for both channel transistors.

AD_n and AD_p are the area of drain diffusion region for both channel transistors.

PD_n and PD_p are the length of drain diffusion region for both channel transistors.

Generally to linearize the depletion capacitances the bottom and sidewall capacitances are multiplied by a constants K_{eq} [1]. The same dimensionless constant K_{eq} is applied to all equivalent junction capacitances, K_{eq} has been evaluated typically for the Northern Telecom CMOS-1B process in Appendix B, giving $K_{eq} = 0.52$.

Hence,

$$C_{eqdbn} = K_{eq}\, CJ_n$$

$$C_{eqdbp} = K_{eq}\, CJ_p$$

$$C_{eqdsn} = K_{eq}\, CJSW_n$$

$$C_{eqdsp} = K_{eq}\, CJSW_p \qquad (2.2.7)$$

Where CJ_n and CJ_p are the zero-bias junction bottom capacitances of junction region per unit area for N- and P- channel transistors.

$CJSW_n$ and $CJSW_p$ are the zero-bias bulk junction sidewall capacitances of junction perimeter per unit length for N- and P-channel transistors.

According to Fig. 2.2.3, the areas of drain diffusion region can be determined by

$$AD_n = W_{dn}\, L_{dn}$$

$$AD_p = W_{dp}\, L_{dp} \qquad (2.2.8)$$

And the perimeters of drain diffusion region can be determined by

$$PD_n = 2(W_{dn} + L_{dn})$$

$$PD_p = 2(W_{dp} + L_{dp}) \qquad (2.2.9)$$

Where W_{dn} and W_{dp} are the drain width of N- and P- channel transistors.

L_{dn} and L_{dp} are the drain length of N- and P- channel transistors.

Moreover, according to Fig. 2.2.3 the gate channel is cross over the diffusion area to share the same width for N- and P-transistors respectively. Substitute Eqns. (2.2.7), (2.2.8), and (2.2.9) into (2.2.6), we have

$$C_{out} = C_{gn}[\gamma_1(1 + \delta_3\delta_5\beta) + \gamma_2(1 + \delta_4\beta) + \gamma_3(1 + \delta_4\delta_5)] \qquad (2.2.10)$$

where

δ_3 is the ratio of bottom equivalent junction capacitance of N- and P-channel transistors.

$$\delta_3 = \frac{CJ_p}{CJ_n} \qquad (2.2.11)$$

δ_4 is the ratio of zero-bias bulk junction sidewall capacitance of N- and P-channel transistors.

$$\delta_4 = \frac{CJSW_p}{CJSW_n} \qquad (2.2.12)$$

δ_5 is the ratio of drain length both P- and N-channel transistors.

$$\delta_5 = \frac{L_{dp}}{L_{dn}} \qquad (2.2.13)$$

Moreover we define γ as

$$\gamma_1 = \frac{K_{eq}\, CJ_n\, W_n\, L_{dn}}{C_{gn}} \qquad (2.2.14)$$

$$\gamma_2 = 2\frac{K_{eq}\, CJSW_n\, W_n}{C_{gn}} \qquad (2.2.15)$$

$$\gamma_3 = 2\frac{K_{eq}\, CJSW_n\, L_{dn}}{C_{gn}} \qquad (2.2.16)$$

δ_3, δ_4, δ_5 and γ_1, γ_2, γ_3 are constants as a functions of the process, design rule; and typical values of these constants for the given Northern Telecom process are given in Table A-2 of Appendix A. The derived equations of input and output capacitance as given by Equations 2.2.3 and 2.2.10 are similar to Kanuma's equation [14] in form, but include more parasitic and peripheral capacitances.

2.2.3 Interstage Capacitance

Now consider the case when an inverter drives another similar inverter. Let the C_c represent the interconnect capacitance. Then C_T, the total interstage capacitance, is represented as a lumped linear peripheral capacitance and is given by the sum of the output capacitance C_{out} of the driving inverter, the input capacitance C_{in} of the driven inverter and the interconnect capacitance C_c. The interconnect capacitance is determined by $C_c = C_x W_c L_c$, where C_x is the capacitance per unit area of the interconnect layer, W_c and L_c are the width and length of the interconnect layer respectively. However, for the buffer design where chain of inverters is used, and each inverter is directly connected to the input of the other inverter, the length of the interconnect is extremely short. Thus this contribution to the overall capacitance is small and thus may be neglected. For instance, in one of our application a layer of Poly runs between two stages only in 6λ. In *NT* CMOS-1B, $C_x = 3.1e{-}4\ pF/\mu m^2$ for Poly equivalent capacitor and $C_c = 0.02\ pF$. Other layers such as metal has even a smaller value of C_c.

Fig. 2.2.4 shows the linear model of interstage capacitance. C_T then is given by

$$C_T = C_{out} + C_{in} \tag{2.2.17}$$

Using Eqns. (2.2.3) and (2.2.10), C_T can be written as

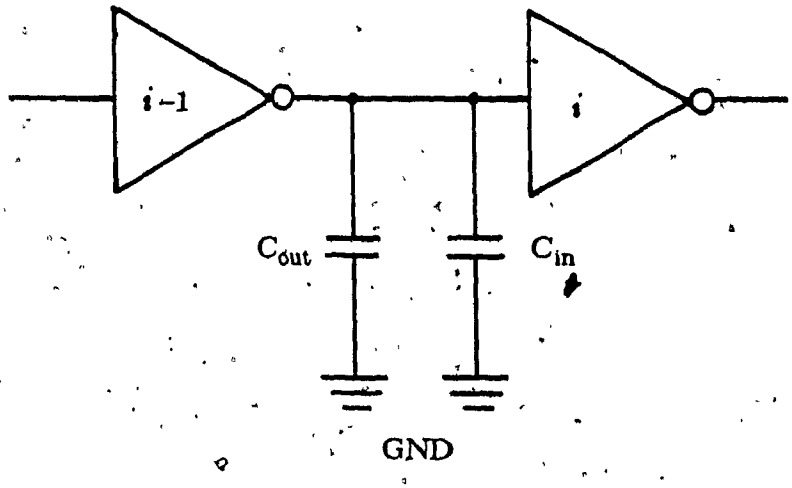


Figure 2.2.4 Cascaded Stage Capacitance

$$C_T = C_g(g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3 + 1) \tag{2.2.18}$$

where $C_g = C_{in}$ in accordance with our previous description and g_1, g_2, and g_3 are given by

where

$$g_1 = \frac{1 + \delta_3\delta_5\beta}{\delta_1 + \delta_2\beta} \tag{2.2.19}$$

$$g_2 = \frac{1 + \delta_4\beta}{\delta_1 + \delta_2\beta} \tag{2.2.20}$$

$$g_3 = \frac{1 + \delta_4\delta_5}{\delta_1 + \delta_2\beta} \tag{2.2.21}$$

Typical values of these process constants for a given process parameters are in the Table A-3 of Appendix A. Also, Table A-5 of Appendix A shows typical values of the input and output capacitances of a single CMOS inverter as a function of the channel width ratio β.

2.3 CMOS Propagation Delay

2.3.1 General Description of CMOS Delay

Propagation delay in the VLSI design is a major problem. We intend to study a typical CMOS inverter delay model in this section. Majority of analytical models for the transient response of CMOS inverter are inaccurate. This is because they use a step input and this is not a realistic input waveform. Consequently errors are introduced into the propagation delay model. In real circuit applications, the input waveform will depend on the fan-out and the driving capability of the preceding stage. In our study, we vary the characteristic waveform by changing the capacitive loading of the preceding inverters relative

to the loading of the inverter under consideration. Thus, the propagation delay will depend on these parameters which is more realistic. Hedenstierna and Jepson [11] have developed a timing model for the CMOS inverter. The model considers the CMOS inverter output response to be an input ramp. They relate the input waveform slope to the delay model by defining an input waveform constant B selected from the consideration of an average slope approximately 70 percent of the characteristic input waveform slope at the 50-percent level and matches the variation of the ramp. Experimental results showed that B approximately matched the input waveform variation within an acceptable error range. Therefore B was assumed to be independent of the input waveform in further time delay analysis. In this section, we adopt the Hedenstierna and Jepson's model and apply it to Northern Telecom CMOS-1B process in order to find the value of B through our own experimental results and study the effect of the input waveform variation.

2.3.2 General Expression of CMOS Delay

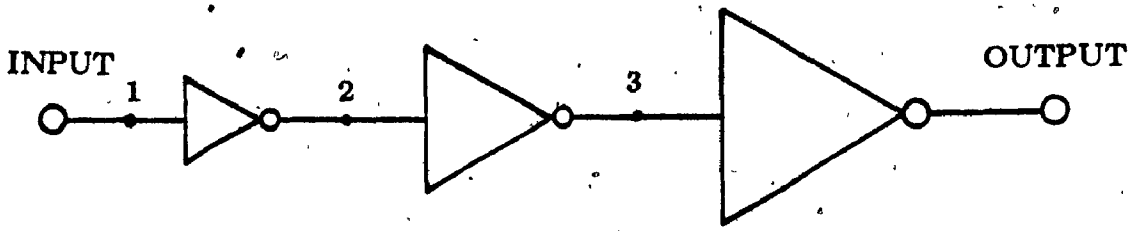
Let us consider the application of a step input waveform to the input of the first inverter of a three-stage inverter buffer as shown in Fig. 2.3.1 (a). The outpout response waveform is shown in Fig. 2.3.1 (b) and is greatly changed in the shape at point 2 and 3. This is due to three factors:

(a) The affect of the input capacitances of the stage under consideration.

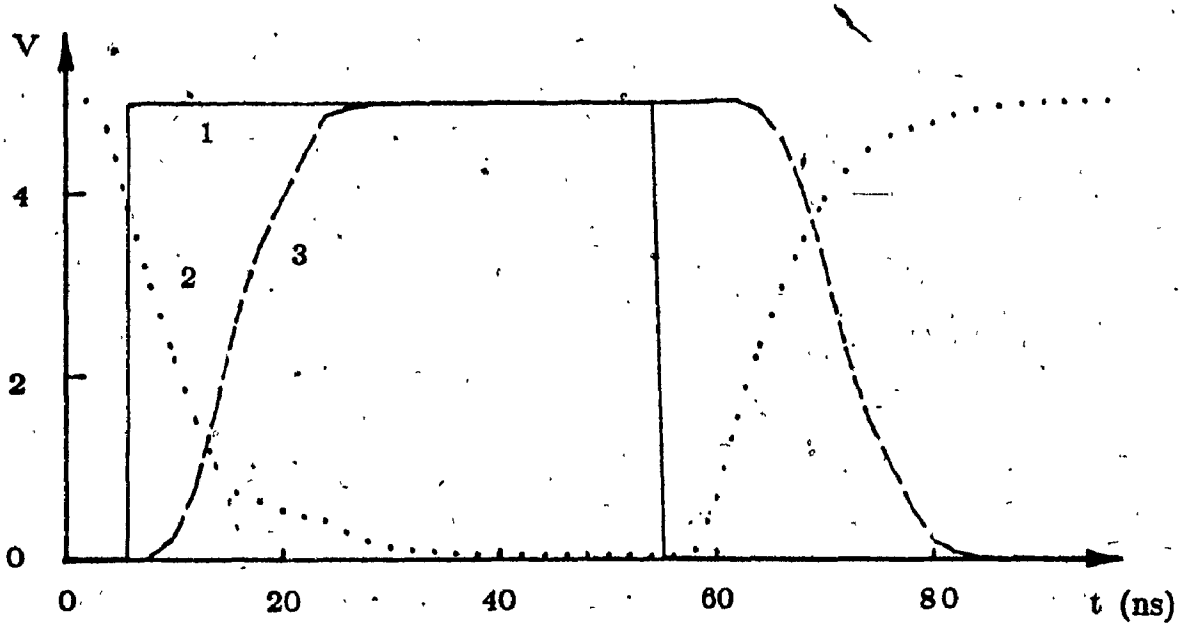
(b) The affect of the output capacitances of the stage under consideration.

(c) The affect of the inherent delay of the inverter itself.

This indicates that the step delay model may not be suitable for the buffer delay estimation, since the input waveform is not a step any more except at the input of the first stage.



(a) Three-stage Buffer



(b) Interstage Waveforms

Figure 2.3.1 Characteristic Waveform Via a Three-stage Inverter Buffer

In the following analysis we will derive the inverter delay model with a ramp input waveform. Let a rising edge be represented as ramp, then the normalized input voltage is written by

$$v_{in} = \begin{cases} 0, & t < 0 \\ s_r\, t, & 0 < t < \tau_r \\ 1, & t > \tau_r \end{cases} \tag{2.3.1}$$

where s_r is the normalized slope of the rising ramp and $\tau_r = 1/s_r$ is the normalized rise time of an inverter.

Similarly the normalized input voltage waveform for the falling edge is represented by

$$v_{in} = \begin{cases} 1, & t < 0 \\ 1 - s_f\, t, & 0 < t < \tau_f \\ 0, & t > \tau_f \end{cases} \tag{2.3.2}$$

where s_f is the slope of the falling ramp and $\tau_f = 1/s_f$ is the fall time of the normalized input voltage. Generally, the ramp waveform is not symmetrical. Fig. 2.3.2 shows a N-stage finite inverter chain, the ramp response of ith stage inverter has been derived in [11] and is given as

$$t_{di} = a\left(\frac{C_L}{K_N}\right)_i + b\left(\frac{C_L}{K_N}\right)_{i-1} \tag{2.3.3}$$

where the subscript $i-1$ and i are the input and output terminal of the ith stage inverter respectively. C_L is interstage load capacitance which is equivalent to interstage capacitance, C_T that was derived in Section 2.2 and K_N is the device transconductance parameter of N-channel transistor and is defined in Appendix B.

The propagation delay of the ith stage may be rewritten by two parts:

1. the step response $t_{ds} = a\,(C_L/K_N)_i$

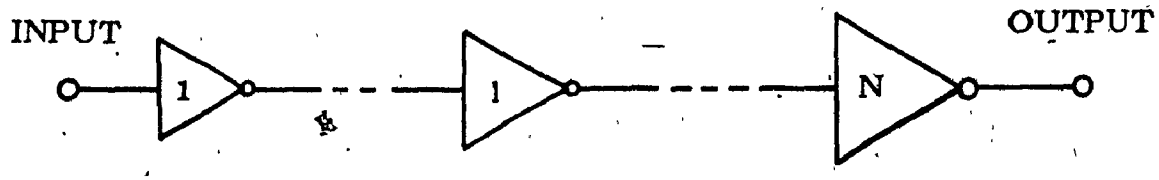


Figure 2.3.2 An N-stage Finite Inverter Chain

and

2. the ramp response $t_{dr} = b\,(C_L/K_N)_{i-1}$

Therefore a is related to the slope of step response and b is related to the slope of ramp response. More detailed discussion of a and b will be given in the next section. The propagation delay of any inverter stage within the buffer of Fig. 2.3.2 can be rewritten as

$$t_d = t_{ds} + t_{dr} \tag{2.3.4}$$

The step response t_{ds} is independent of the inherent input capacitances but dependent on the physical characteristics (e.g. layout, design rule etc.) and the output load capacitance. The ramp response t_{dr} is quite dependent on the input capacitance.

2.3.3 Matching the Ramp

As mentioned in the previous section, a is part of the slope of step response and b is part of the slope of ramp response. Moreover they are functions of β, the channel width ratio. Applying the result in [11], a is given by

$$a = \frac{1}{2}\left[A_N + \frac{A_P}{\beta\frac{\mu_p}{\mu_n}}\right] \tag{2.3.5}$$

where A_N and A_P are functions of the process constants for the N- and P-channel devices, and the supply voltage. From well known step-response propagation delay equation [1], A_N and A_P are given as

$$A_N = \frac{1}{V_{DD}(1-n)}\left[\frac{2n}{1-n} + \ln\left(\frac{2(1-n)-v_o}{v_o}\right)\right] \tag{2.3.6a}$$

$$A_P = \frac{1}{V_{DD}(1+p)}\left[\frac{-2p}{1+p} + \ln\left(\frac{2(1+p)-v_o}{v_o}\right)\right] \tag{2.3.6b}$$

where $v_o = V_{out}/V_{DD}$ is the normalized output voltage, $n = V_{TN}/V_{DD}$ and $p = V_{TP}/V_{DD}$ are the normalized threshold voltages of N- and P-channel transistors.

And similarly, the b 's expression is resulted as

$$b = \frac{1}{2}\left[B_P + \frac{B_N}{\beta\frac{\mu_p}{\mu_n}}\right] \tag{2.3.7}$$

where B_P and B_N are functions of input waveform for P- and N-channel transistors. They are developed from Hedenstierna's equation and represented respectively by

$$B_{Ni-1} = \frac{1+2n}{6}\left(\frac{K_P}{C_L}\right)_{i-1}\left(\frac{1}{s_r}\right)_i \tag{2.3.8a}$$

$$B_{Pi-1} = \frac{1-2p}{6}\left(\frac{K_N}{C_L}\right)_{i-1}\left(\frac{1}{s_f}\right)_i \tag{2.3.8b}$$

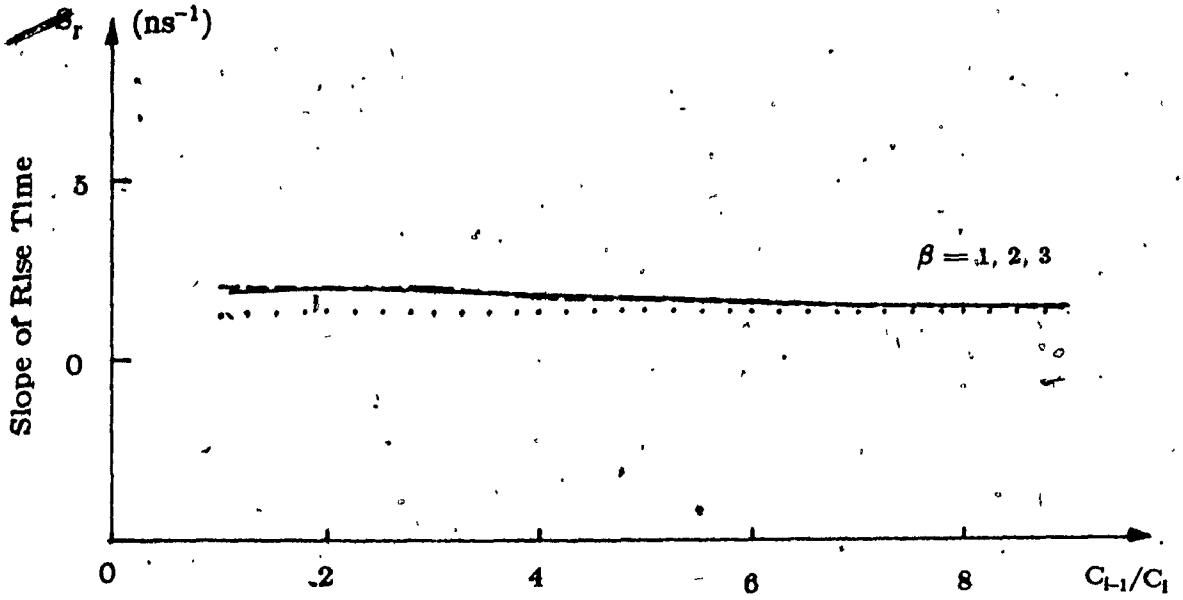
where the subscript $i-1$ and i are the nodes of the i th stage inverter's input and output terminals, and K_P and K_N are the device transconductance parameters for P- and N-channel transistors. Yielding Equations (2.3.3), (2.3.5), (2.3.6a), (2.3.6b), (2.3.7), (2.3.8a), and (2.3.8b) where K_P is replaced by $K_P = \beta(\mu_p/\mu_n)K_N$ and μ_p and μ_n are the surface mobility of P- and N-channel transistors, the values of μ_p and μ_n are given in Appendix A. In this context the propagation delay through the i th stage inverter may be written as

$$t_{di} = \gamma_4(\tau_f + \tau_r)_i + a\left(\frac{C_L}{K_N}\right)_i \tag{2.3.9}$$

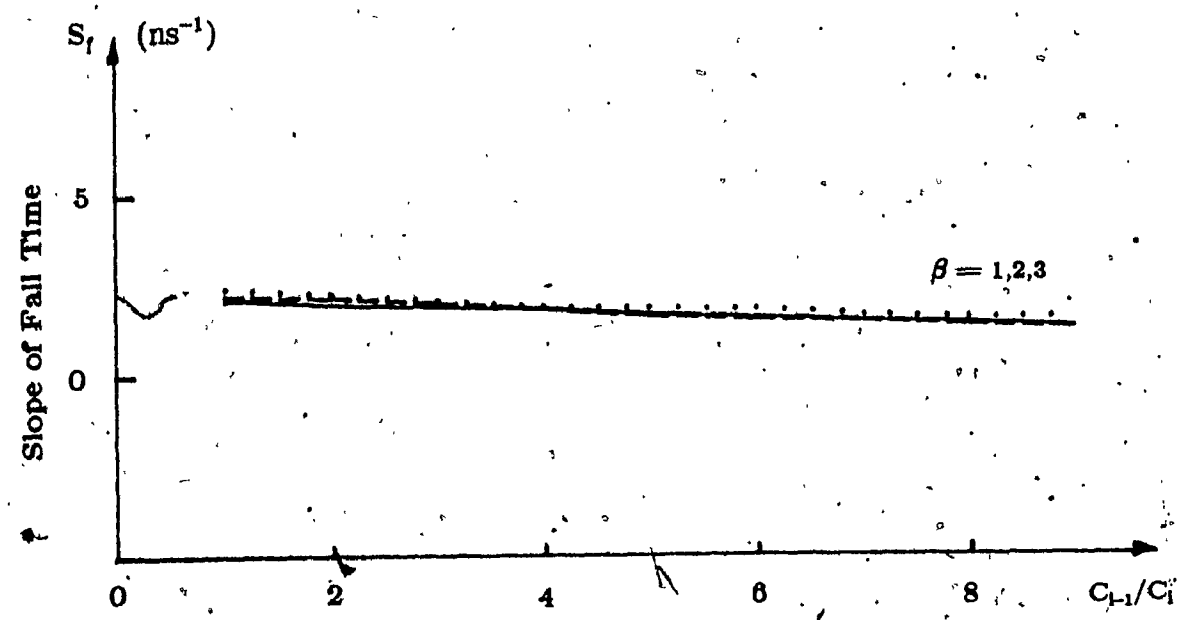
where $\gamma_4 = (1-2p)/12 = (1+2n)/12$ is process constant. Notice that this equation is different from the equation of $t_d = (\tau_r + \tau_f)/4$, which is to average the rise and fall times to determine the delay. Equation (2.3.9) is more accurate,

since the step response is theoretically calculated by the term of $a\,(C_L/K_N)$, and the ramp response part is obtained by τ_r and τ_f at the output terminal of the inverter. Also, it is interesting to notice that the input capacitance C_{Li-1} has disappeared from the equation (2.3.9). This is true, because the loading capacitance only affects the output waveform shape which comes from preceding stage, and not the input waveform itself [11]. Regarding the general case of hand analysis, the rise and fall times of the output waveform is not obtained by direct reading. Therefore, we must derive an equation in terms of loading factor; in another words, we estimate the propagation delay through the loading factor changes instead of measuring the results of the response from the rise and fall times. The loading factor is an integer multiple of the fan-in and fan-out of the inverter loading under consideration. Once the input slope is converted to the loading factor, the propagation delay is then a function of the fan-out and the driving capability of the stage itself, as in the step-response model, but also a function of the fan-out and the driving capability of the preceding stage. To do so, first of all, the input waveform dependent function B (i.e. B_N or B_P) has to be determined.

Fig. 2.3.3 (a) and (b) are plots of the variation of an inverter normalized output rise and fall slopes with input capacitive loading and the P- and N-channel width ratio β as a parameter. This indicates that both the output rise and fall slopes of a single inverter is almost constant for variation of input load and a constant output load. Hence we assume that the output slope is independent of input loads. Indicating that $(B_N)_{i-1}$ and $(B_P)_{i-1}$ are independent of slope, therefore the terms of K_p/C_L and K_n/C_L may be removed from the equations (2.3.8a) and (2.3.8b). In order to match the time response between step and ramp input waveform, the averaged time slope is chosen from experimental results, which is approximately 63% of the step waveform slope and then this



(a) Rise Time Slope



(b) Fall Time Slope

Figure 2.3.3 Normalized Output Slopes with Input Capacitance Loading Variation

averaged slope ($= 1.52\ ns^{-1}$ for normalized voltage V_{in}/V_{DD}) for both of rise and fall time is substituted into equation (2.3.8a) and (2.3.8b) respectively, we have the input waveform constant $B_N = B_P = 0.149$ which is the best fit to the input ramp.

Since we used practical process parameters Miller effect was neglected. Similar to B_N and B_P the A_N and A_P have been adjusted to experimental value 0.2 instead of 0.308 for theoretical value of $n = -p = 0.18$ and $V_{DD} = 5$ volts at the 50-percent level. Thus, a and b can be determined by Eqns. (2.3.5) and (2.3.7) respectively, the typical values of a and b are given in the Table A-3 of Appendix A.

Now we look into the loading capacitance C_{Li} at node i. If a inverter derives n same uniform size inverters, the loading capacitance C_{Li} is represented by

$$C_{Li} = (C_{out})_{i-1} + n(C_{in})_i \tag{2.3.10}$$

Notice that the subscript $i-1$ and i are $i-1$th and ith stage respectively. By substituting Equation (2.3.10) into Equation (2.3.3) the propagation delay through the ith stage inverter is then given by

$$t_d = a\left(\frac{C_{out} + nC_{in}}{K_N}\right)_i + b\left(\frac{C_{out} + mC_{in}}{K_N}\right)_{i-1} \tag{2.3.11}$$

This is a propagation delay for an inverter with a fan-out of n identical inverters and a fan-out of m identical inverters in the preceding stage. According to Eqn. (2.2.18), Eqn. (2.3.10) can be written as

$$C_L = C_g(g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3 + n) \tag{2.3.12}$$

furthermore using Eqns. (2.3.10), (2.3.11) and (2.3.12) the propagation delay may now be written by

$$t_d = \frac{C_g}{K_N} a \left[(n + g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3) + \frac{b}{a}(m + g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3) \right] \quad (2.3.13)$$

the propagation delay can be divided into three different delays.

The intrinsic delay

$$t_{di} = \frac{C_g}{K_N}(a + b)(g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3) \quad (2.3.14)$$

The fan-out dependent delay

$$t_{do} = \frac{C_g}{K_N} an \quad (2.3.15)$$

The input load dependent delay

$$t_{din} = \frac{C_g}{K_N} bm \quad (2.3.16)$$

2.3.4 SPICE Simulation

To validate the derived model, SPICE simulation was performed on inverters designed with Northern Telecom's CMOS-1B process parameters which are listed in Table A-1 of Appendix A.

Simulation 1

Fig. 2.3.4 shows a chain of three-stage inverters. The propagation delay for the middle inverter is to be evaluated. Since the input step waveform is supplied by SPICE and will not be affected by input capacitances variation, an additional inverter is located in the first position of the chain between the signal source and the studied inverter's input terminal. The third inverter which is connected to

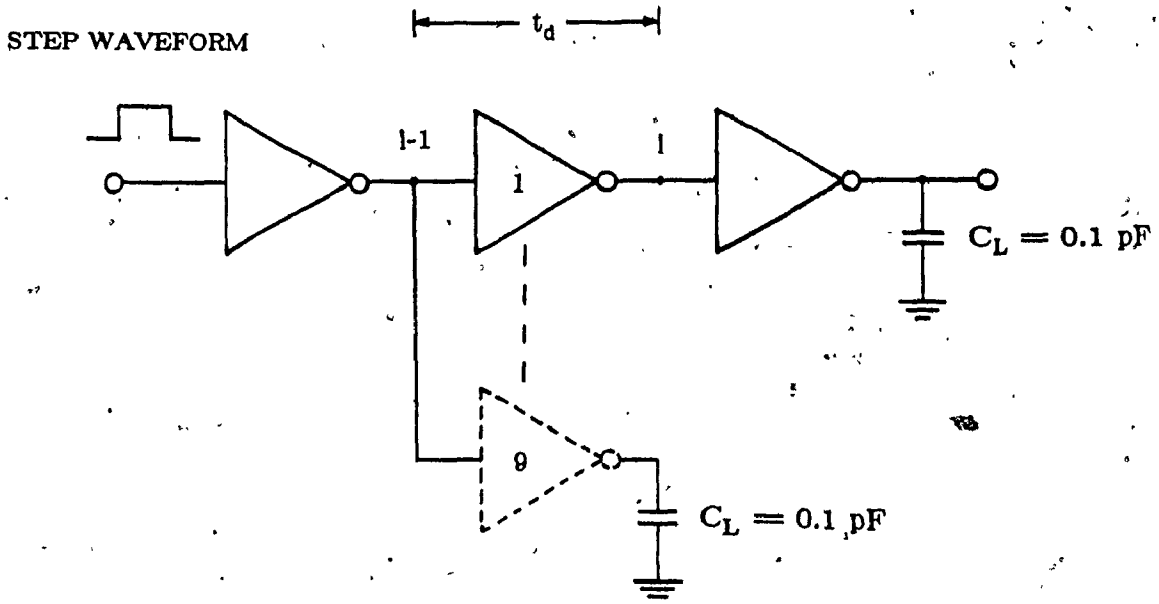


Figure 2.3.4 Simulation Circuit for the Input Loading Study

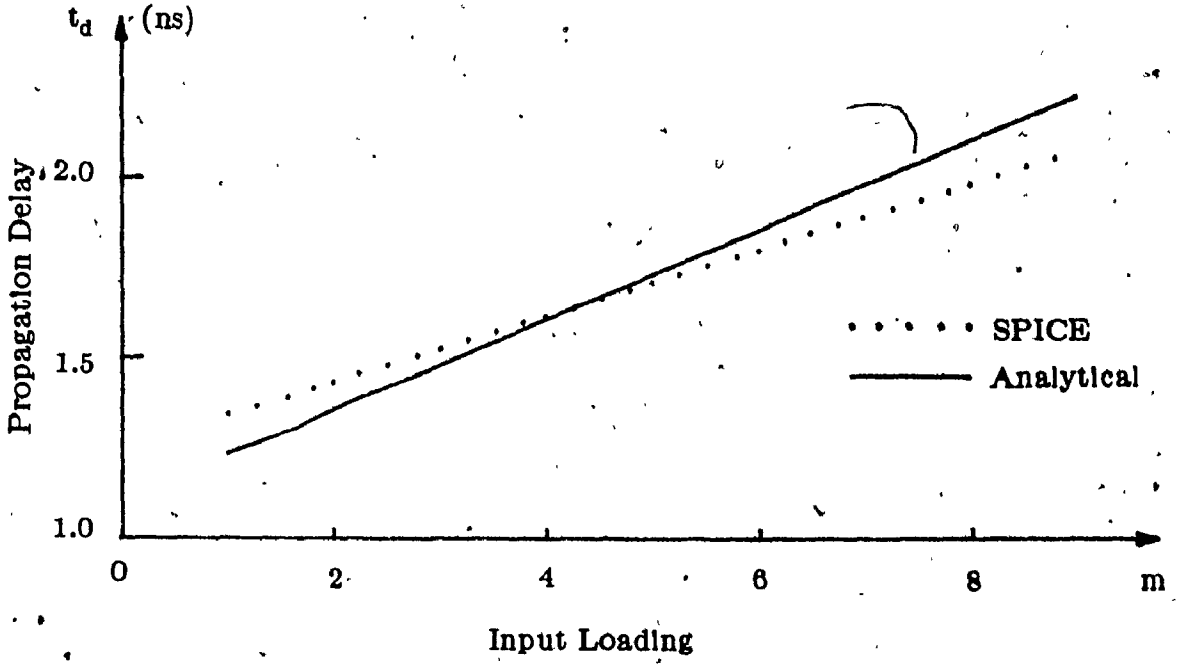
the studied inverter's output terminal is an output loading inverter for the studied inverter. All inverters have the same uniform size. Here the uniform size has been chosen as minimum size inverter. For instance, for the CMOS-1B 5 micron technology the minimum size inverter is characterized by the width of N-channel transistor equal to 5 μm. The change in input load is studied by addition of similar inverters at node $i-1$ of the studied inverter's input terminal. Up to nine inverters were added including the studied inverter itself. The purpose of increasing the number of inverters is to achieve the change in capacitive loading. Therefore the delay at the node i is ramp response according to the input waveform at node $i-1$.

Figs. 2.3.5 (a), (b), (c) illustrate comparison of theoretical analysis and SPICE simulation results for variation of the propagation delay, with loading and β. This indicates that, the average error between SPICE simulation and analytical result for all simulated results are expected to be less than 10%.

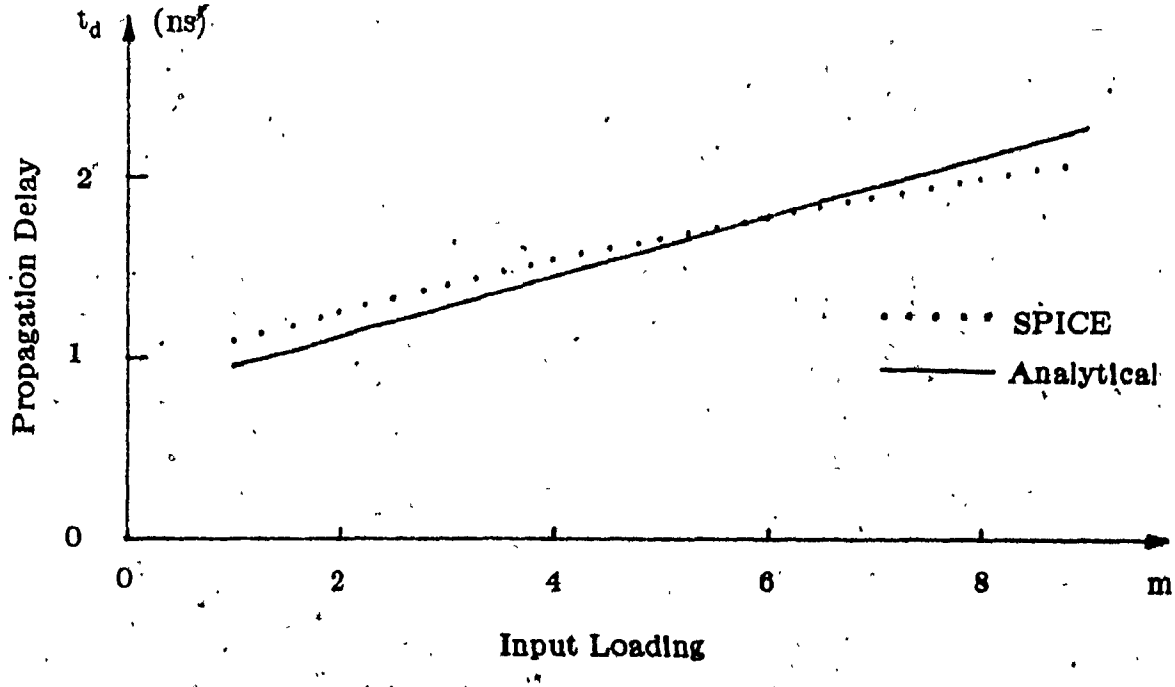
Simulation 2

This simulation has been made to observe the time delay variation in response to increase in output loads. This is equivalent to observing the waveform variation at node $i-1$ in simulation 1. The simulation circuit is shown in Fig. 2.3.6. Figs. 2.3.7 (a), (b) and (c) illustrate the comparison of analytical and SPICE simulation results for the propagation delay variation due to purely output loads and channel width ratio β with constant input load. one fan-in was chosen in this plot, at 50-percent level. Also indicating that, the averaged errors between SPICE simulation and analytical results for all curves are expected to be less than 8%.

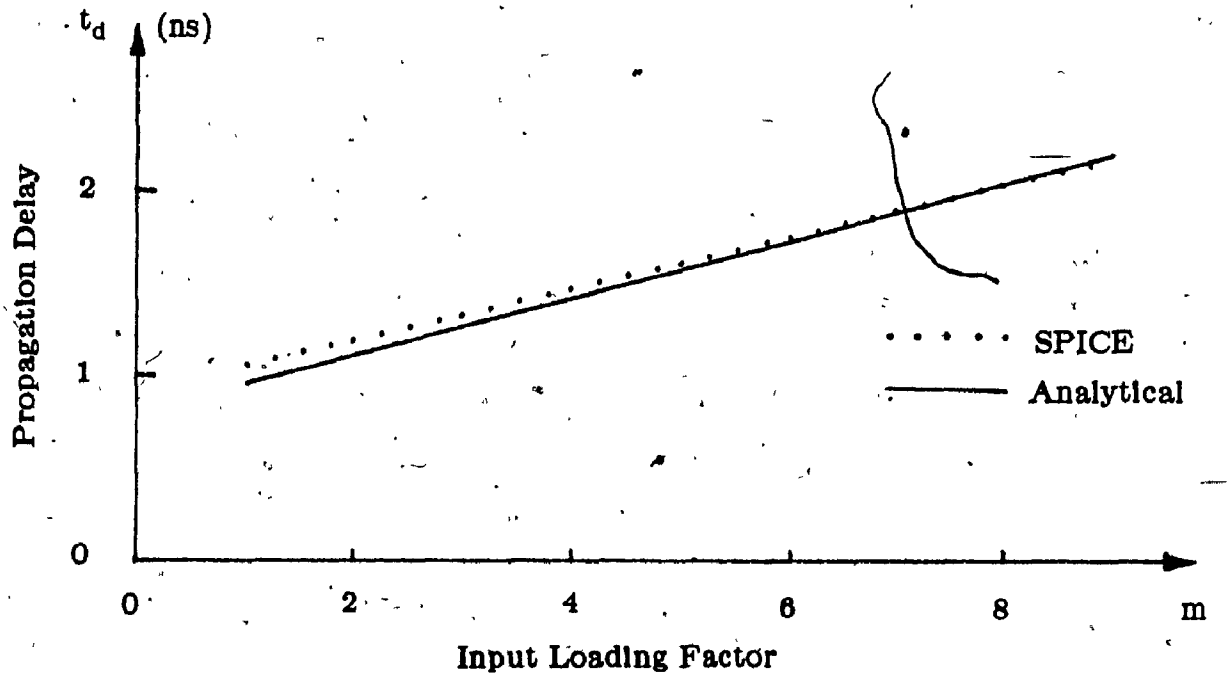
From the theoretical analysis and simulation results, we confirm that the expression describing the propagation delay time is good enough to estimate the



(a) Channel Width Ratio $\beta = 1$



(b) Channel Width Ratio $\beta = 2$



(c) Channel Width Ratio $\beta = 3$

Figure 2.3.5 Variation of Propagation Time with Input Loading Factor

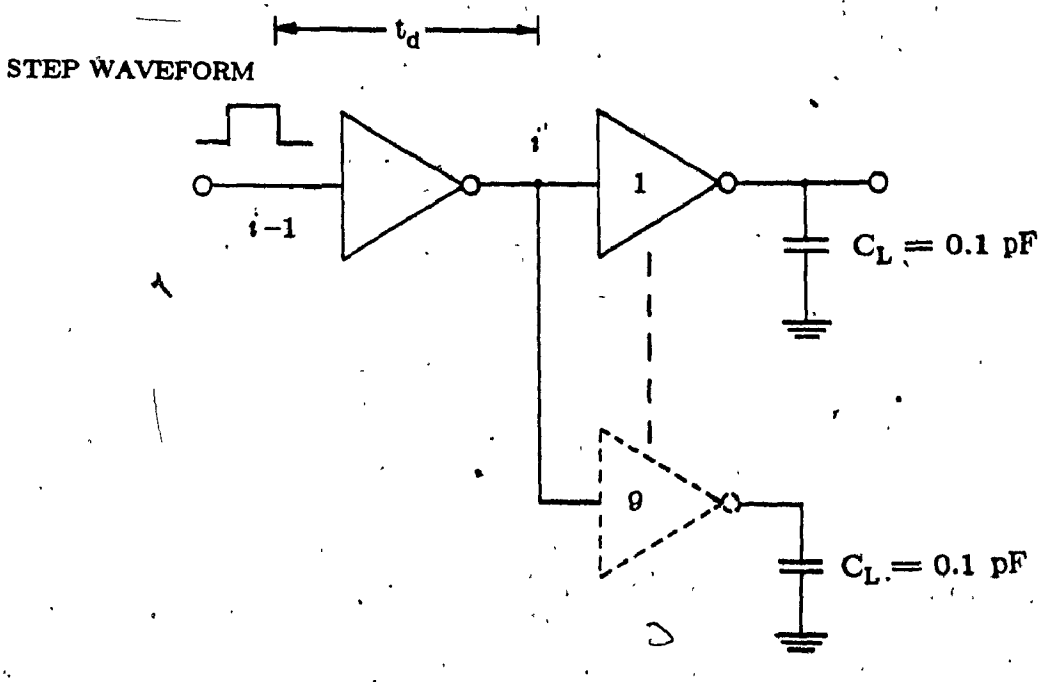
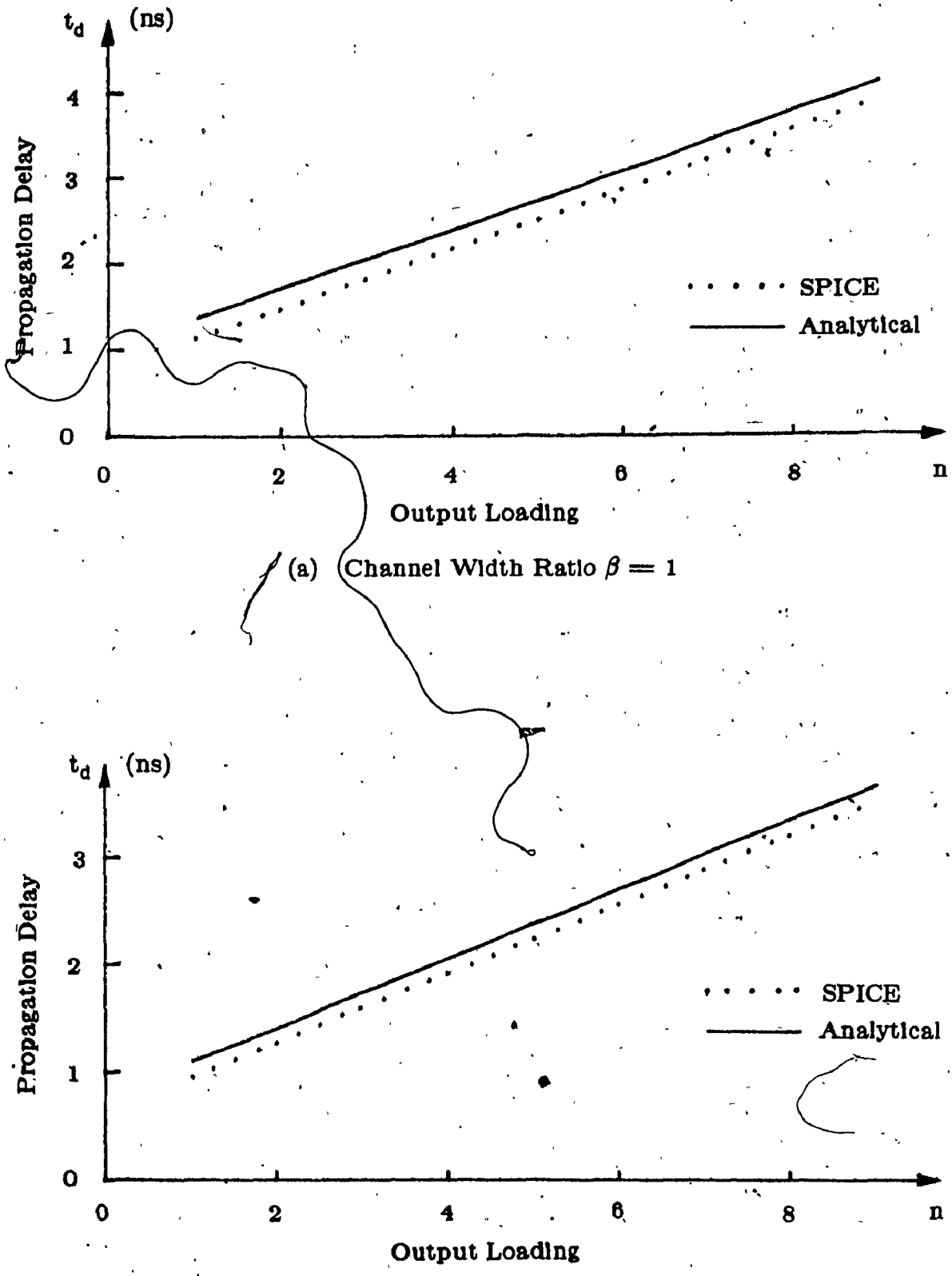
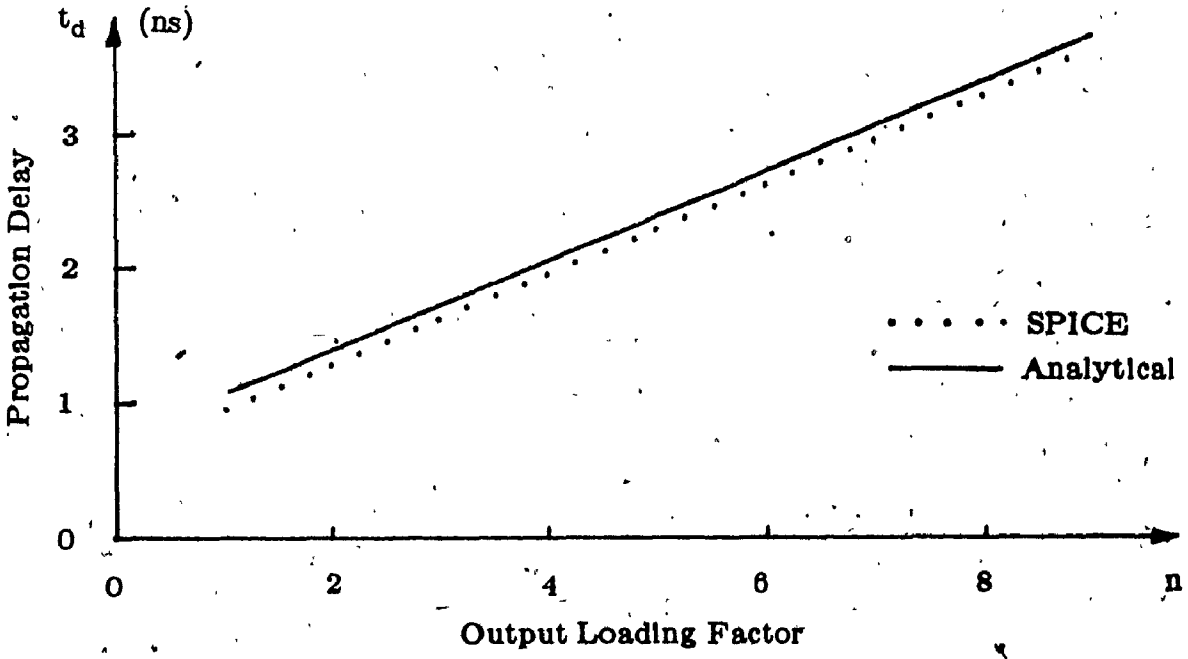


Figure 2.3.6 Simulation Circuit for the Output Loading Study



(a) Channel Width Ratio $\beta = 1$

(b) Channel Width Ratio $\beta = 2$.



(c) Channel Width Ratio $\beta = 3$

Figure 2.3.7 Variation of Propagation Time with output Loading Factor

time delay for a single CMOS inverter, and for all CMOS circuits characterized by pull up and pull down structures. The programs for simulation 1 and 2 are given in Appendix C.

During SPICE simulation, a problem had been noticed which introduces reading errors of results from the simulation. This error is inherently introduced whenever the time step in SPICE was changed. Fig. 2.3.8 shows the error difference when using 0.005 *ns* and 0.01 *ns* time step increments. This can introduce an error of up to 4.95%, when the time step was varied. Naturally this error can reduce the degree of confidence in the developed model.

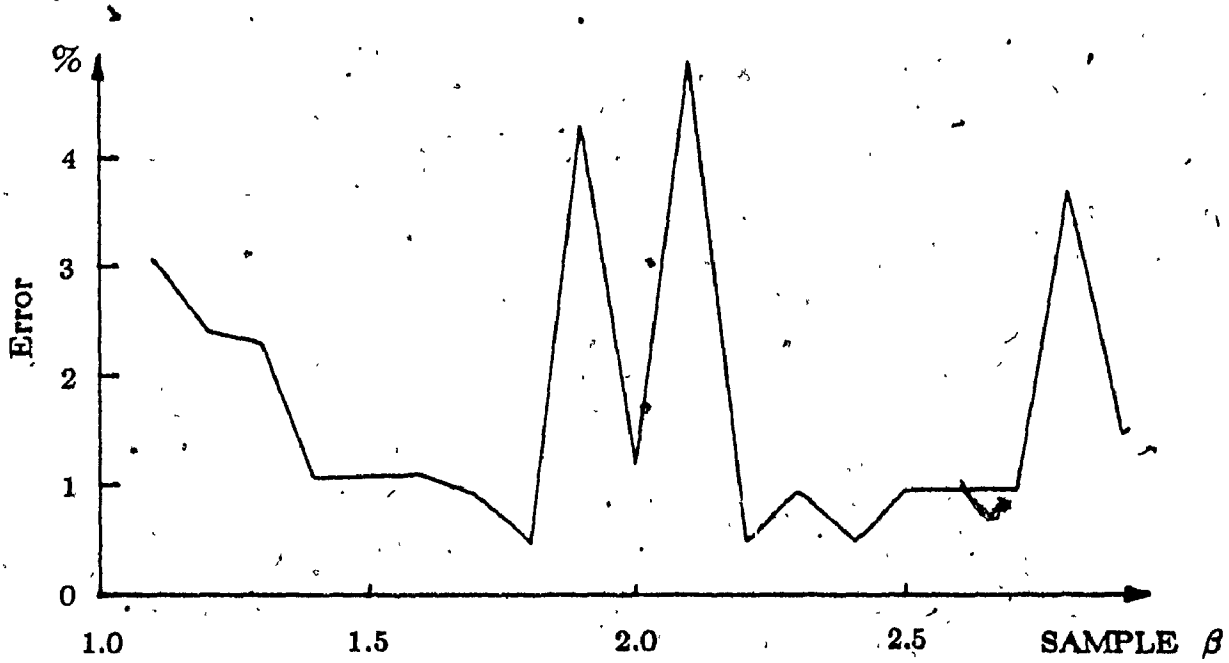


Figure 2.3.8 Statistics of Error in Misreading

CHAPTER 3

SPEED OPTIMIZATION OF CMOS DEVICE

3.1 Introduction

In this chapter, methods of minimizing propagation delay for CMOS circuits are described.

Parameters affecting delay are studied and two parameters affecting delay are singled out, these are β, the channel width ratio and the scaling factor. An analytical expression of optimum channel width ratio β is described in Section 3.2 that gives minimum delay for a single CMOS inverter.

In Section 3.3.1 we review an analytical expression for a N-stage CMOS buffer and the buffer's delay is related to the general process constants $g\gamma$ and scaling factor.

For buffer driving large capacitive loads, we optimize the buffer for minimum delay in Section 3.3.2. The main result of the buffer optimization is that, when the ramp timing model is taken into account, the optimum scaling factor is increased from theoretical 2.7 [14] to the range 3-5.

3.2 Optimum β for Minimum Delay

To obtain minimum delay, the analytical expression of CMOS propagation delay given by Eqn. (2.3.13) is differentiated with respect to β, recalling that C_g, a, b and g are all functions of β. For the given uniform size N-channel transistor, the optimum β is then given by

$$\beta_o = \left[\frac{\mu_n}{\mu_p} \frac{A_P(\gamma_1+\gamma_2+\gamma_3+\gamma_3+\gamma_3\delta_5+n\,\delta_1)+B_N(\gamma_1+\gamma_2+\gamma_3\delta_5+m\,\delta_1)}{A_N(\gamma_1\delta_3+\gamma_2\delta_4+n\,\delta_2)+B_P(\gamma_1\delta_3+\gamma_2\delta_4+m\,\delta_2)} \right]^{1/2} \quad (3.2.1.)$$

optimum β_o is a function of fan-in m, fan-out n and other process and layout parameters. For a fixed process and layout parameters, four cases are studied for different values of m and n. Case 1: fan-in m equals fan-out n and varied. Case 2: fan-in m is fixed at a certain number and fan-out is varied. Case 3: fan-in is varied and fan-out is fixed at a certain number. Case 4: Both m and n are varied independently.

Fig. 3.2.1 shows the relationship between optimum β_o and fan-in and fan-out loads. Curve 1 $m = n$ and varied; curve 2 $m = 1$ and n is varied; curve 3 $n = 1$ and m is varied. Plots of 3.2.1 clearly indicate that β_o decreases as the input or output load increases.

Propagation delays are also obtained as a function of channel width β by SPICE simulation and the results are plotted in Fig. 3.2.2. It is a excellent agreement according to curve 1 at $m = n = 1$ in Fig. 3.2.1. the curves show that the minimum delay is around at $\beta = 2.4$, for the Northern Telecom's process parameters and the $n = -p = 0.18$, $A_N = A_P = 0.2$, $B_N = B_P = 0.149$, $V_{DD} = 5$ volts and at 50-percent level analysis and $m = n = 1$. The other point to notice is that variation of the propagation delay around β_o is not vary high.

3.3 CMOS Buffer

3.3.1 Time Analytical Expression for N-stage Finite CMOS Buffer

In VLSI circuit design, it is common that various size buffers are required to drive various loads. Some of their loads are large such as clock driver or output

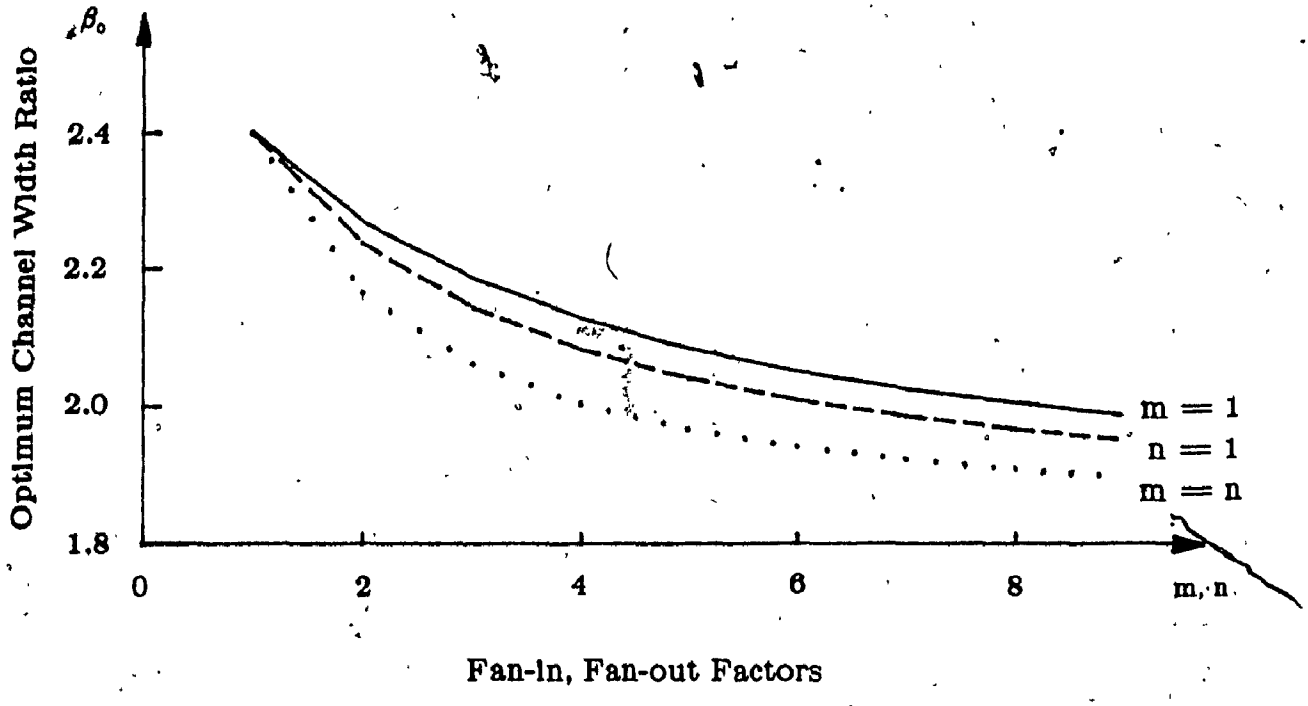


Figure 3.2.1 Optimum β with Fan-in and Fan-out Variation

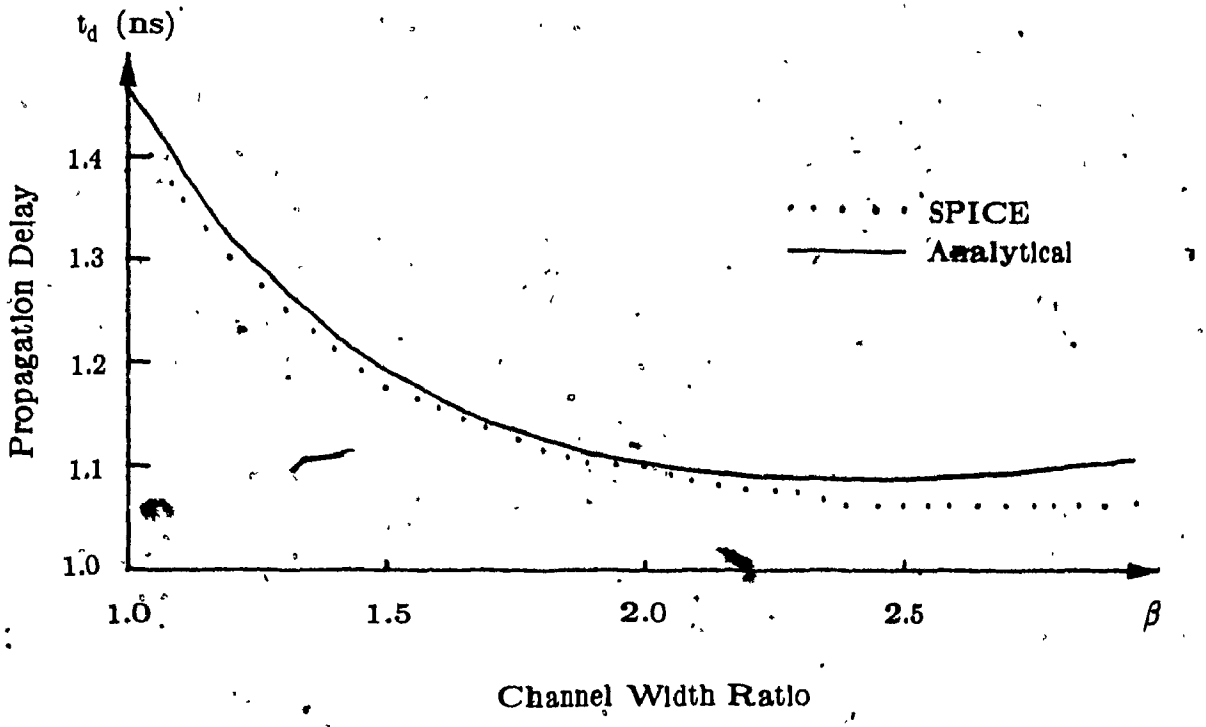


Figure 3.2.2 Comparison of Propagation Time with Channel Width Ratio β Variation

pad driver. It is a common practice to design a chain of inverters to replace a large buffer. This is achieved by gradually increasing geometrical size of each stage inverter to form a buffer such that at each stage, inverter is driving a slightly larger inverter. Fig. 3.3.1 shows an N-stage ideal CMOS inverter buffer. C_0 is the input capacitance of the buffer, which is the lumped capacitance of the first inverter's gate capacitance C_{go} and other external capacitance if other loading is applicable. C_N is output capacitance of buffer, which is a lumped capacitance of the last inverter's output capacitance C_{Nout} and the output external capacitance C_{load}, and $C_{i-1}(i = 1, 2, ..., N+1)$ is the interstage capacitance at node $i-1$. A scaling factor S is defined by the increment of the gate capacitance as following

$$S = \frac{C_{g1}}{C_{go}} = \cdots = \frac{C_{gi}}{C_{gi-1}} = \cdots = \frac{C_{load}}{C_{gN-1}} \tag{3.3.1}$$

or the i th inverter's gate capacitance can be represented as

$$C_{gi} = S^i C_{go} \tag{3.3.2}$$

Meanwhile, the device transconductance parameter K_i of the respective stage for $L = L_0 = \cdots = L_N$ are also held by the equation (2.3.1) as

$$S = \frac{K_1}{K_o} = \cdots = \frac{K_i}{K_{i-1}} = \cdots = \frac{K_N}{K_{N-1}} \tag{3.3.3}$$

If all the inverters have the same value of channel width ratio β, Then Eqn. (3.3.1) and (3.3.3) can be written as

$$\left(\frac{C_g}{K}\right)_0 = \cdots = \left(\frac{C_g}{K}\right)_{i-1} = \cdots = \left(\frac{C_g}{K}\right)_{N-1} \tag{3.3.4}$$

According to Fig. 3.3.1 and Eqn. (2.2.18), we look at the output terminal of the i th stage at node i, the interstage capacitance C_i can be written as

$$C_i = C_{gi-1}[(g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_{i-1} + S] \tag{3.3.5}$$

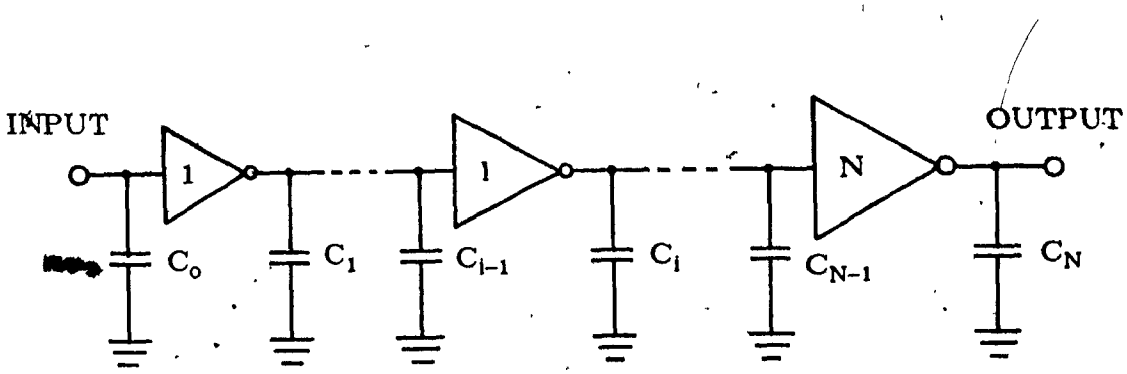


Figure 3.3.1 An N-stage Ideal CMOS Inverter Buffer

the average propagation delay through the i th stage inverter is now written as follows using Eqn. (2.3.13)

$$t_{di} = \frac{C_g}{K_N} a \left\{ [S + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_{i-1}] + \frac{b}{a}[S + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_{i-1}] \right\}$$

$$= \tau_o [S + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_{i-1}] \qquad (3.3.6)$$

where $\tau_o = (C_g/K_N)(a + b)$, and C_g/K_N is the constant ratio of inverter input gate capacitance to N-channel transistor for any of the inverters in the buffer chain. For the minimum size inverter the unit delay τ_o as a function of β is given in table A-4 of appendix A.

Since the inverter chain consists of N stages, the total delay can be written as

$$t_{Total} = \tau_o N [S + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o] \qquad (3.3.7)$$

Let the total fan-out of the buffer or (the ratio between the output external load capacitance C_{load} of the last (Nth) inverter and the gate capacitance C_{go} of the first inverter) be defined as

$$Y = \frac{C_{load}}{C_{go}} \qquad (3.3.8)$$

Meanwhile, the relationship between C_{load} and C_{go} is given by

$$C_{load} = S^N C_{go} \qquad (3.3.9)$$

Yielding Eqns. [3.3.8] and [3.3.9] the buffer's fan-out Y's expression in terms of S and N is given by

$$Y = S^N \qquad (3.3.10)$$

To eliminate N in Eqn. (3.3.7), using $N = \ln Y / \ln S$, the buffer delay can be rewritten as

$$t_{Total} = \tau_o \ln Y [S + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o] / \ln S \qquad (3.3.11)$$

In the analysis to follow, the input capacitance is always modeled as an output capacitance of the minimum size inverter and it has the same value of β as the inverter's inside the buffer. furthermore, the first inverter is to be assumed as minimum size inverter in analysis as well.

3.3.2 Speed Optimization for CMOS Buffer

So far the analytical expression of time delay for a buffer has been derived, and now we look into buffer speed optimization. The optimum performance parameter that minimizes overall delay, the optimum scaling factor S_o or alternatively, the optimum number of inverter stages N_o. In order to find optimum S_o for minimum buffer delay, we differentiate t_{Total} with respect to S.

$$\frac{d(t_{Total})}{dS} = \frac{C_g}{K_n}(a+b)\ln Y \left[\frac{1}{\ln S} - \frac{S + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o}{S(\ln S)^2}\right] \qquad (3.3.12)$$

the optimum S_o then can be determined by

$$S_o = e^{[S_o + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o] / S_o} \qquad (3.3.13)$$

As we see that $g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3 = 0$ yields $S_o = e$, which is according to mead and conway [15] who have done this optimization neglecting the intrinsic load capacitance of the inverter. We also notice that the optimum scaling factor S_o in the Eqn. (3.3.13) is independent of capacitance. The buffer propagation delay as a function of scaling factor S with β as parameter ($\beta = 1, 2, 3$) is shown in Fig. 3.3.2. Notice that the optimum scaling factor S is slightly varied in

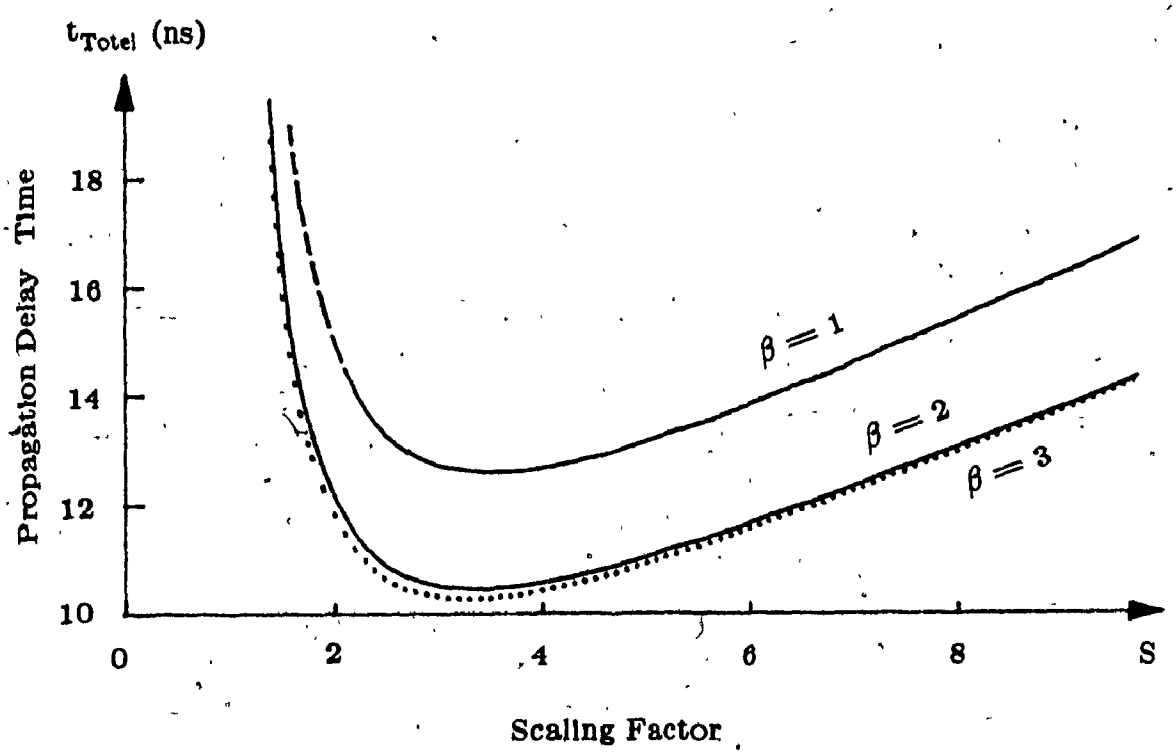


Figure 3.3.2 Variation of Propagation Time with Scaling Factor

the range of 3.27-3.49 with respect to β variation. Hence, a 0.22 difference may be neglected and an averaged value 3.37 of optimum scaling factor may be reasonable to the minimum propagation delay time. Once the scaling factor is determined, the optimum number of inverter stages N_o for the buffer can be found i.e.

$$N_o = \frac{\ln Y}{\ln S_o} \tag{3.3.14}$$

However, the number of inverter stages N can only be an integer number. We have to check whether $N = r$ or $N = r + 1$, where $r < N_o < r + 1$, gives the shortest buffer delay.

Using $S = Y^{1/N}$ to eliminate S in Eqn. (3.3.7), the delay can be rewritten as

$$t_{Total} = \tau_o N [Y^{1/N} + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o] \tag{3.3.15}$$

the optimum number of inverters is then $N = r$ if

$$r [Y^{1/r} + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o]$$

$$< (r+1)[Y^{1/(r+1)} + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o] \tag{3.3.16}$$

and $N = r + 1$ if

$$r [Y^{1/r} + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o]$$

$$> (r+1)[Y^{1/(r+1)} + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o] \tag{3.3.17}$$

However, our analysis showed that the propagation delay would be varied, not more than 3 percent if $N = r$ was chosen instead of $N = r + 1$ in the second case, but considerable savings in area may be achieved as discussed in the *Chapter 5*. The above algorithm will be applied in our buffer design implementation at *Chapter 6*, and since the chosen number of inverter stages may not be exactly equal to the optimum number yielding the optimum scaling factor

S_o, the actual scaling factor should be adjusted to

$$S = Y^{1/N} \tag{3.3.18}$$

and always an integer number of N is selected. The propagation delay through an N-stage inverter buffer for a given process against number of inverter stages is plotted in Fig. 3.3.3.

Both Figures 3.3.2 and 3.3.3 show that an optimum scaling factor exists that minimizes the propagation delay. The figures show this in the range of load capacitance chosen from $1pF$ to $10pF$. However, the optimum S and N are related and they follow the exponential relationship. We will also consider other objective functions (*e.g.* *Area*, *Power*, *PowerDelay*, *AreaTime*, $AreaTime^2$, *etc.*), and determine the optimum scaling factor. These analysis are performed in *Chapter 4*.

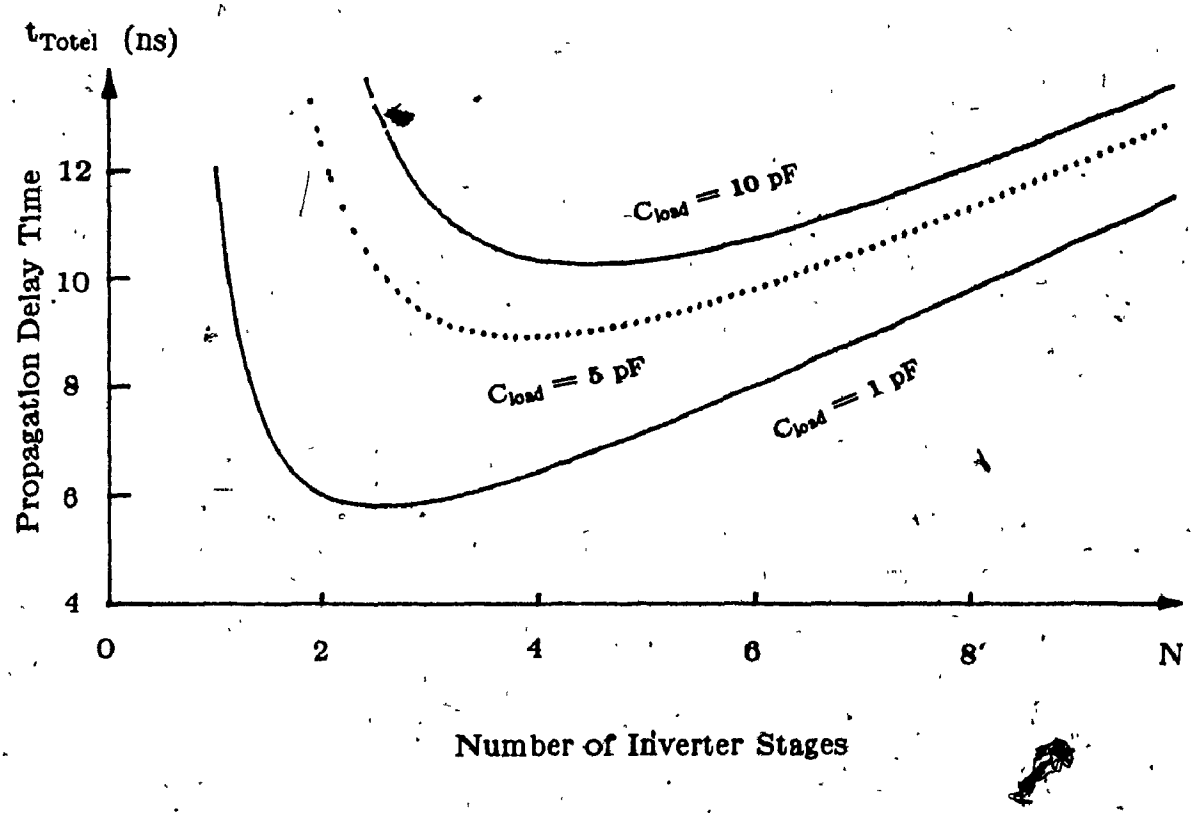


Figure 3.3.3 Variation of Buffer Propagation Time with Number of Inverter Stages

CHAPTER 4

POWER CONSUMPTION OF CMOS DEVICE

4.1 Introduction

The general description of the power dissipation of CMOS device is given in this chapter. Some discussions of the power dissipation have been presented for CMOS device [2], [3], [11] and [16]-[19]. However, we develop the power dissipation model and expressions based on analytical results which have been derived from previous chapters. There are two components that establish the amount of power dissipated in a CMOS device, $P = P_s + P_d$, These are :

1. P_s is a static dissipation due to leakage current.
2. P_d is a dynamic dissipation which has two components too, $P_d = P_{sc} + P_t$.
   i. P_{sc} is a short-circuit power dissipation.
   ii. P_t is a transient power dissipation.

where P_{sc} and P_t are due to

a) switching transient current.

b) charging and discharging of load capacitances.

In Section 4.2, the static power dissipation of CMOS inverter is described. Dynamic power dissipation is discussed in Section 4.3. Section 4.4 presents an expression for the estimation of power dissipation of an N-stage finite CMOS inverter buffer.

4.2 CMOS Static Power Dissipation

Fig. 4.2.1 shows a complementary CMOS inverter. If the input is at "0" state, the associated N-channel transistor is "OFF" and the P-channel transistor is "ON". The output voltage is V_{DD} or logic "1". When the input is at "1", the associated N-channel transistor is biased "ON" and the P-channel transistor is "OFF". The output voltage is at GND level or logic "0" state. Note that one of the transistors is always "OFF" when the gate is in either of these logic states. Since no current flows into the gate terminal, and there is no D.C. current path from V_{DD} to GND, the resultant quiescent (steady state) current, and hence power P_s, is zero.

However, there is some small static dissipation due to reverse bias leakage between diffusion regions and the bulk. some parasitic diodes can be modeled between all source, drain diffusion regions and P-well diffusion to bulk. Fig. 4.2.2 shows a simple model that describes the parasitic diodes for a CMOS inverter in order to have an understanding of the leakage involved in the device. In this model, diode D is a parasitic diode between p-well to bulk. Since parasitic diodes are reverse biased, only their leakage current contributes to static power dissipation, the leakage current is described by the diode equation

$$I = IS\,(e^{\frac{qV}{KT}} - 1) \qquad (4.2.1)$$

where IS = Bulk junction saturation current

V = voltage across the diode (Volts)

q = electronic charge (Coulomb)

k = Boltzmams's constant (Joule/Kelvin)

T = temperature (Kelvin)

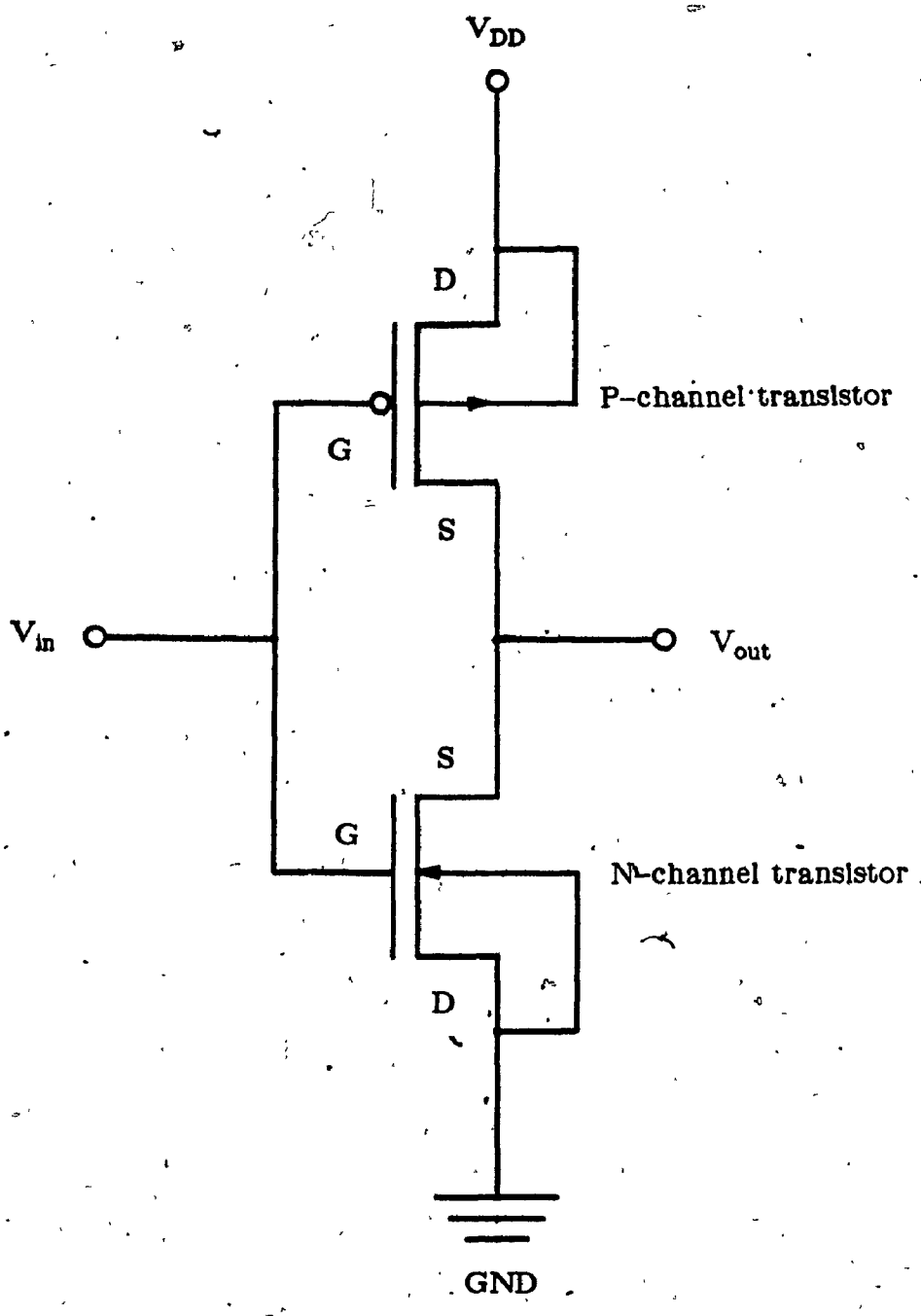


Figure 4.2.1 A Complementary CMOS Inverter

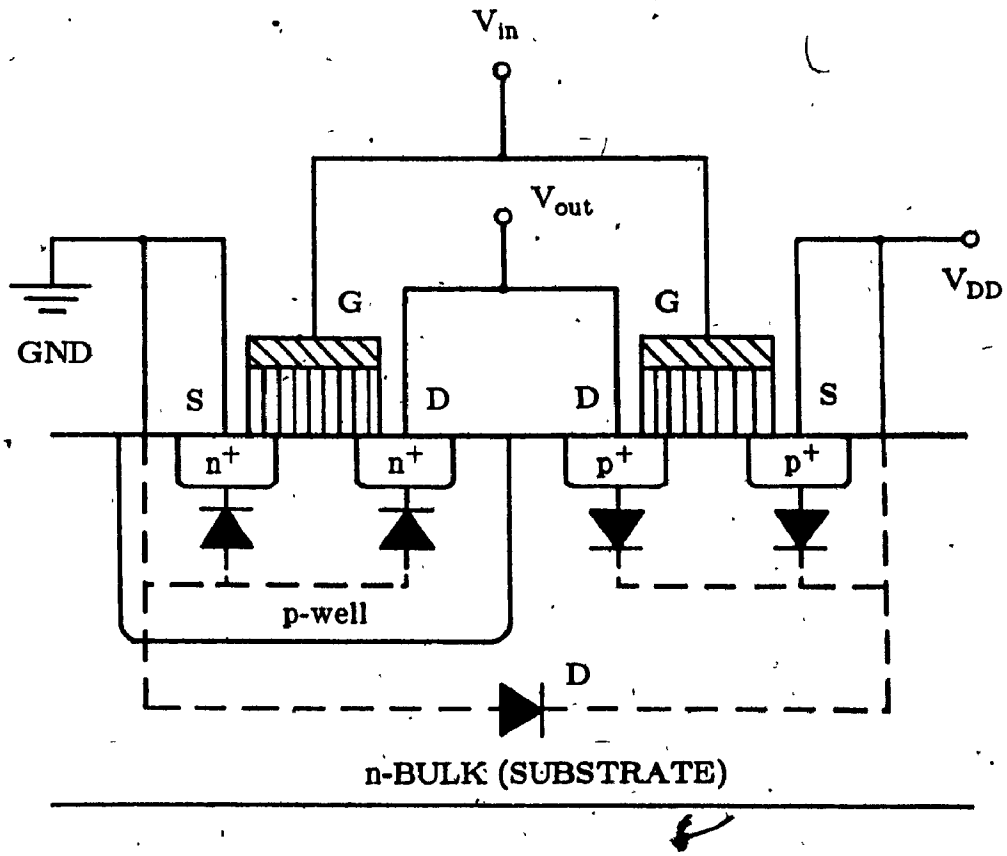


Figure 4.2.2 A Simple Model of Parasitic Diodes

For the given NT's process parameters $IS = 1e-14\ (A)$. In addition, unless special precautions are taken, the leakage current across the surface of the device will be in the order of $1e-9\ (A)$ per gate at room temperature, the leakage current increases as the applied voltage is increased. The static power dissipation is the product of the device leakage current and the supply voltage..

$$P_s = IV_{DD} \tag{4.2.2}$$

Typical static power dissipation due to leakage for an inverter operating at 5 volts is about 5 nano-watts. For an N-stage CMOS inverter buffer, the static power dissipation can be represented as

$$P_s = V_{DD} \sum_{i=1}^{i=N} I_i \tag{4.2.3}$$

where N is the number of CMOS inverter's stage and I_i is the leakage current of i th stage inverter. However, for a CMOS circuits, seitz [20] had commented that there is no static power dissipation; there is only dynamic power dissipation during transition in operation. The matter of the static power dissipation may be neglected toward our further analysis as long as it's a small percentage of power dissipation.

4.3 CMOS Dynamic Power Dissipation

The current through the CMOS inverter, varies depending on each transistor and what region of operation the transistor is operating at. For the P-channel transistor, the drain to source current I_p is represented by

$$-I_p = \begin{cases} K_p[(V_{in}-V_{DD}-V_{thp})(V_{out}-V_{DD})-(V_{out}-V_{DD})^2/2], & 0 \le V_{in} < V_{out}+V_{thp} \\ K_p(V_{in}-V_{DD}-V_{thp})^2/2, & V_{out}+V_{thp} \le V_{in} < V_{DD}-V_{thp} \\ 0, & V_{in} \ge V_{DD}-V_{thp} \end{cases} \tag{4.3.1a}$$

and the drain to source current for N-channel transistor

$$I_n = \begin{cases} 0, & 0 \le V_{in} < V_{thn} \\ K_n (V_{in} - V_{thn})^2/2, & V_{thn} \le V_{in} < V_{out} + V_{thn} \\ K_n [(V_{in} - V_{thn}) V_{out} - V_{out}^2/2], & V_{out} + V_{thn} \le V_{in} \le V_{DD} \end{cases} \quad (4.3.1b)$$

Moreover, we normalize voltages with respect to V_{DD} as was the case in previous Chapters, Eqns. (4.3.1a) and (4.3.1b) can be rewritten as

$$-i_p = \begin{cases} K_p V_{DD}^2 [(v_{in} - 1 - p)(v_o - 1) - (v_o - 1)^2/2], & 0 \le v_{in} < v_o + p \\ K_p V_{DD}^2 (v_{in} - 1 - p)^2/2, & v_o + p \le v_{in} < 1 - p \\ 0, & v_{in} \ge 1 - p \end{cases} \quad (4.3.2a)$$

and similarly the drain to source current for N-channel transistor is given as

$$i_n = \begin{cases} 0, & 0 \le v_{in} < n \\ K_n V_{DD}^2 (v_{in} - n)^2/2, & n \le v_{in} < v_o + n \\ K_n V_{DD}^2 [(v_{in} - n) v_o - v_o^2/2], & v_o + n \le v_{in} \le 1 \end{cases} \quad (4.3.2b)$$

If we load the inverter output with a capacitive loading C_L, and a trapezoidal input voltage with slow ramp, Fig. 4.3.1 shows the behaviors of input voltage and current waveform. During transition from either state "0" to "1" or, alternatively, from state "1" to "0", regarding the ramp waveform, there is a time period in which both N- and P-channel transistors conduct, resulting a short-circuit to flow from voltage supply V_{DD} to ground GND according to Fig. 4.2.1, then the power dissipation of the circuit is considered separately for that time period.

Definition 1.

The short-circuit power dissipation P_{sc} is that energy that is dissipated only by the occurrence of the edge of input signal waveform.

Definition 2.

The transient power dissipation P_t is that energy that is dissipated only by the occurrence of the square wave without edging affect of the input signal waveform. Then, the dynamic dissipation of the circuit consists of two components:

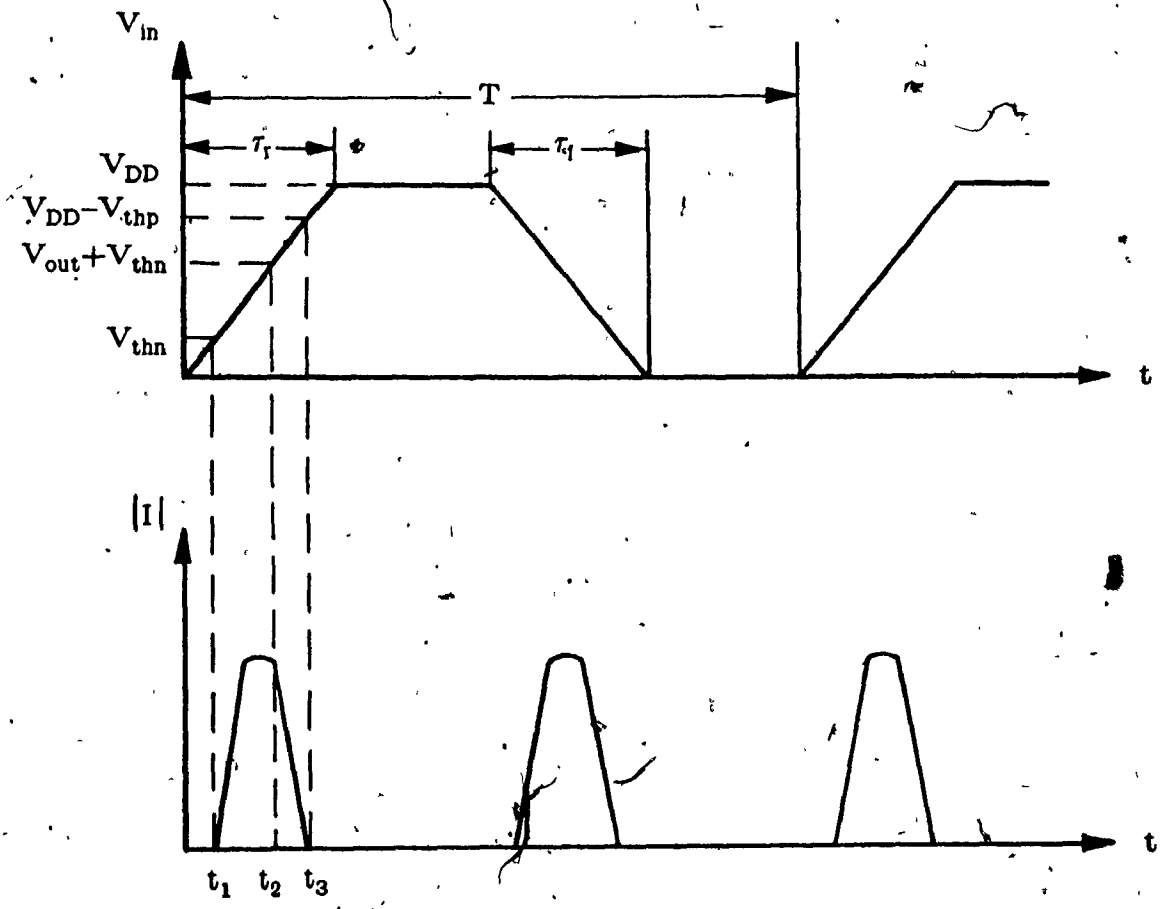


Figure 4.3.1 Behaviors of Input Voltage and Current Waveform

1. the short-circuit power dissipation.

2. the transient power dissipation.

The short-circuit current and the transient power dissipation strongly depends on the inverter design (process parameters). But the transient power dissipation for the case of single inverter with a loading capacitor is apartly from contributions due to parasitic output capacitances, such as junction capacitances. For the case of inverter buffer except the last stage, the transient power dissipation of each inverter strongly depends on the inverter design too. Our analysis will be given detailed discussion in the following sections.

4.3.1 Short-circuit Power Dissipation

Two analytical expressions for calculation of short-circuit power dissipation of a CMOS inverter with load and without load have been developed by Hedenstierna [11] and Veendrick [18]. In the following analysis we derive our own expression for the short-circuit power dissipation with reference to Fig. 4.3.1.

In the region of the input voltage, $V_{in} = s_r\, t$, and as long as the P-channel transistor is saturated the differential equation for the current i_p may be written as

$$i_p(t) = C_L \frac{dv_o}{dt} = -\frac{K_p}{2} V_{DD}^2 (s_r\, t - 1 - p)^2 \qquad (4.3.3)$$

where $i_p(t)$ is the normalized drain to source current for P-channel transistor as a function of time t, C_L is the capacitive load and with the initial condition $v_o = 1$ for $s_r\, t = 1 + p$, intergration yields

$$v_o = 1 - \frac{K_p\, V_{DD}^2}{6 s_r\, C_L} (s_r\, t - 1 - p)^3 \qquad (4.3.4)$$

The N-channel transistor is conducting as long as $n \leq v_{in} \leq 1$ and $0 \leq v_{in} \leq 1 + p$ for P-channel transistor, so the integral region is $(n, 1+p)$ for both transistors conducting. The input voltage v_{nl}, when the N-channel transistor is entering the linear region, is determined by $v_{nl} = v_o + n$.

The short-circuit power dissipation for the current through the N-channel transistor per transition is represented as

$$P_{sc} = V_{DD} \int_{t_1}^{t_3} i_n(t)\,dt \tag{4.3.5}$$

where $i_n(t)$ is the normalized drain to source current for N-channel transistors as a function of t. $t_1 = n/s_r$ and $t_3 = (1+p)/s_r$ are the time period for both transistors conducting. But, for the time period of t_1 and t_3 according to Fig. 4.3.1, the current I_n has two expressions for the saturated and linear regions which is separated by an intermediate time $t_2 = V_{nl}/s_r$, the time for which the N-channel transistor enters the linear region. Yielding Eqns. (4.3.5) and (4.3.2b), the short-circuit power dissipation is written as

$$P_{sc} = K_N V_{DD}^3 \left\{ \frac{1}{2} \int_{n/s_r}^{v_{nl}/s_r} (v_{in} - n)^2 dt + \int_{v_{nl}/s_r}^{(1+p)/s_r} [(v_{in} - n)v_o - \frac{1}{2} v_o^2]\,dt \right\} \tag{4.3.6}$$

where the N-channel transistor is saturated in the first integral and is linear in the second integral. Substitute the expression of v_o given by Eqn. (4.3.4) into Eqn. (4.3.6) and integrate, yields

$$P_{sc} = C_L V_{DD} d_n \left\{ \frac{d_p}{24}(v_{nl}-n-1)(v_{nl}-1-p)^4 + \frac{1}{2}\left[(1+p-n)^2-(v_{nl}-n)^2\right] + \frac{1}{6}(v_{nl}-n)^3 - \frac{d_p}{120}(v_{nl}-1-p)^5 + \frac{d_p^2}{504}(v_{nl}-1-p)^7 \right.$$

$$+ \frac{1}{2}(v_{nl} - 1 - p)\Big\} \qquad (4.3.7)$$

where $d_n = K_N V_{DD}^2 / s_r C_L$, $d_p = K_P V_{DD}^2 / s_r C_L$ are process dependent constants and C_L is interstage load capacitance, substituting $K_P = \beta(\mu_p / \mu_n)$ and the relationship between d_p and d_n is given by

$$d_p = \beta \frac{\mu_p}{\mu_n} d_n \qquad (4.3.8)$$

The short-circuit power dissipation for the current through the P-channel transistor while the load capacitor is discharged by the N-channel transistor is similar method, the P-channel stays in saturation region and the N-channel transistor moves from saturation to linear. The expression had been derived by Hedenstierna [11], but some mistakes were apparently made in the calculation of the integral. A correct expression of short-circuit power dissipation per transition is given by

$$P_{sc} = C_L V_{DD} d_p \left[\frac{d_n}{24}(1 - v_{pl} + p)(v_{pl} - n)^4 - \frac{d_n}{120}(v_{nl} - n)^5 \right.$$

$$\left. - \frac{d_n^2}{504}(v_{pl} - n)^7 + \frac{1}{6}(1 - v_{pl} + p)^3 \right] \qquad (4.3.9)$$

where $v_{pl} = v_o + p$ is the input voltage, when the P-channel transistor is leaving the linear region. Eqns. (4.3.7) and (4.3.9) show a general situation, since the input signal has different rise and fall time while asymmetrical inverter is involved as part of buffer.

The short-circuit power dissipation for the negative ramp input voltage is the same as those given in Eqns. (4.3.7) and (4.3.8) just reverse the sign of input voltage. Therefore the short-circuit power dissipation per clock cycle of time period is double the short-circuit power dissipation per transition, and can be

written by

$$P_c = 2\frac{1}{T}P_{sc} \tag{4.3.10}$$

where $f = 1/T$ the frequency of switching, resulting in

$$P_c = 2P_{sc}\, f \tag{4.3.11}$$

The analytical expression for calculation of an average short-circuit power dissipation per switching in a geometrically symmetrical CMOS inverter and with the same slope for rise and fall time input voltage without capacitance load is given by Veendrick [18]

$$P_{sc} = \frac{K_N V_{DD}^3}{12s}(1 - 2n)^3 f \tag{4.3.12}$$

Notice that Eqns. (4.3.7), (4.3.9) and (4.3.12) , depend on the design parameters of K_N, K_P, β and slope s of the rise and fall times of input voltage. Also the short-circuit power dissipation is proportional to the switching frequency.

4.3.2 Transient Power Dissipation

The transient power dissipation is modelled by assuming the rise and fall time of the step input is much less than the repetition period or ideally say that the rise and fall time are zero according to definition 2. The average transient power P_t, dissipated during switching for a standard square wave input voltage V_{in}, having a repetition frequency of $f = 1 / T$, is given by Weste and Eshraghian [2]

$$P_t = \frac{1}{T}\int_0^{T/2} i_n(t) V_{out}\, dt + \frac{1}{T}\int_{T/2}^{T} i_p(t)(V_{DD} - V_{out})dt \tag{4.3.13}$$

where the energy is dissipated during the positive step wave in the first integral and during the negative step wave in the second integral, substituting

$i_n(t) = -i_p(t) = C_L\, dV_{out}/dt$ into Eqn. (4.3.13) and the transient power dissipation per clock cycle of time period can be written as

$$P_t = \frac{C_L}{T}\int_0^{V_{DD}} V_{out}\, dV_{out} + \frac{C_L}{T}\int_{V_{DD}}^{0} (V_{DD} - V_{out})\, d(V_{DD} - V_{out})$$

$$= \frac{C_L V_{DD}^2}{T} \qquad (4.3.14)$$

with $f = 1/T$ resulting in

$$P_t = C_L\, V_{DD}^2\, f \qquad (4.3.15)$$

Thus for a repetitive step input the average transient power that is dissipated is proportional to the energy required to charge and discharge the circuit capacitance. The important factor to be noticed here, is that Eqn. (4.3.15) shows transient power to be proportional to switching frequency but depends on the design parameters, since C_L is interstage load capacitance. For a single inverter, this is partly from contributions due to parasitic output capacitance of it's junction.

Fig. 4.3.2 shows comparison of transient and short-circuit power dissipation per clock cycle as a function of the inverter's load capacitance and the rise time of input voltage as a parameter. The short-circuit dissipation is not negligible at the range of small values of load capacitance, it's even bigger than transient dissipation. While the load capacitance $C_{load} = 0$, the short-circuit dissipation reaches to peak and this agrees to Veendrick's result [17] of the short-circuit current behavior. We evaluated our expression experimentally by taking a slow rise time $\tau = 15\ ns$, and plotting the short-circuit power as a function of inverter load capacitances. As it can be seen the short-circuit power dissipation is exponential decaying while the load capacitances increases almost linearly with the load. For instance, the short-circuit power dissipation only is a fraction ($<8.5\%$) of the dynamic power dissipation, when the load capacitance is

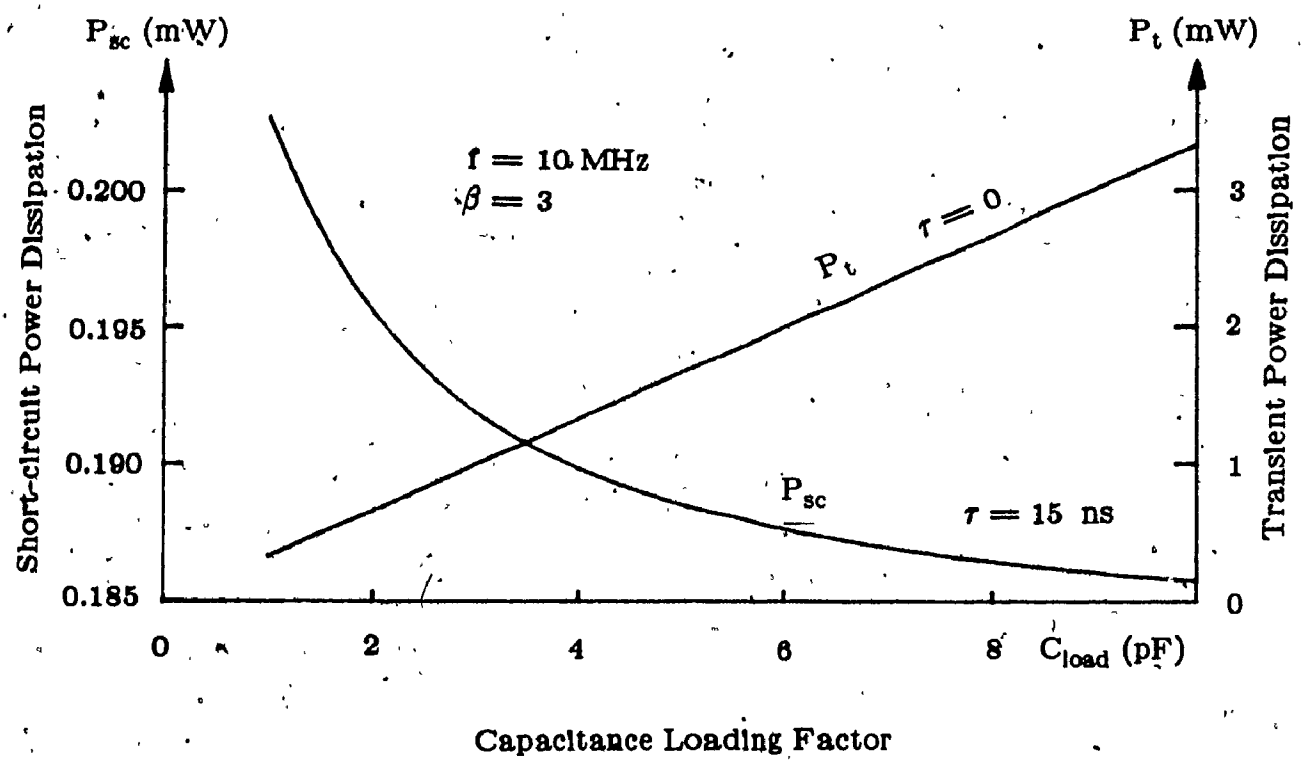


Figure 4.3.2 Comparison of Transient and Short-circuit Power Dissipation

increased to 8 pF.

4.4 Dynamic Power of CMOS Buffer

Theoretically, the dynamic power dissipation per clock cycle of an N-stage finite CMOS inverter buffer is proportional to the sum of the individual inverter's dissipation. However, in order to find the optimum power dissipation we will neglect the short-circuit power dissipation in the estimation of buffer power consumption. Since the short-circuit energy dissipation is directly proportional to the average short-circuit current and this current is faintly contributing to the charge and discharge of the load capacitor, if the buffer drives a large load capacitance. It is an acceptable approximation as long as the short-circuit power dissipation is smaller to the power needed to charge the capacitor. For simplicity we will derive an expression of buffer's power consumption which only consists of transient dissipation.

To determine the dynamic power dissipation per clock cycle of an N-stage finite CMOS inverter buffer, we take the summation of each inverter's dissipation, such as

$$P_B = \sum_{i=1}^{N} P_{di} \tag{4.4.1}$$

where P_{di} is the ith stage inverter's dynamic dissipation, thus using Eqns. (4.4.1), (4.3.15), (3.3.2), and (2.2.8), we have

$$P_B = V_{DD}^2 f \sum_{i=1}^{N} C_{Li}$$

$$= V_{DD}^2 f \sum_{i=1}^{N} S^{i-1} C_{go} [S + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o]$$

$$= \frac{1 - S^N}{1 - S}[S + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o]V_{DD}^2 f C_{go} \qquad (4.4.2)$$

where S is the scaling factor of the buffer, C_{Li} is an interstage capacitance at node i, and N is the number of inverter stage. The load capacitance $C_{load} = S^N C_{go}$ is inserted into Eqn. (4.4.2) and the expression of dynamic power dissipation per clock cycle for the N-stage finite CMOS inverter buffer can be written as

$$P_B = \frac{1 - S^N}{(1 - S)S^N}[S + (g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3)_o]V_{DD}^2 f C_{load} \qquad (4.4.3)$$

Fig. 4.4.1 and Fig. 4.4.2 plot the dynamic power dissipation per clock cycle of a CMOS buffer as a function of the scaling factor S and the number of the inverter stages N respectively at working frequency $f = 10\ MHz$, the ratio of channel width $\beta = 3$ and load capacitance $C_{load} = 10\ pF$.

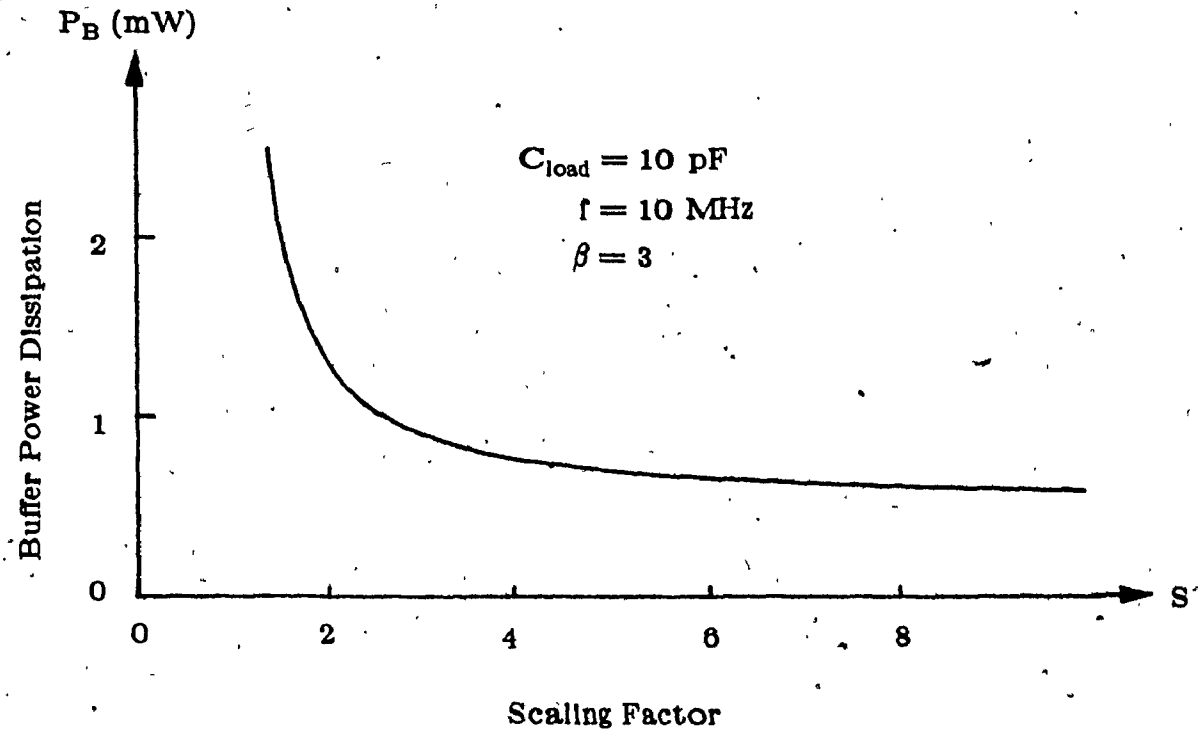


Figure 4.4.1 Power Dissipation as a Function of Scaling Factor

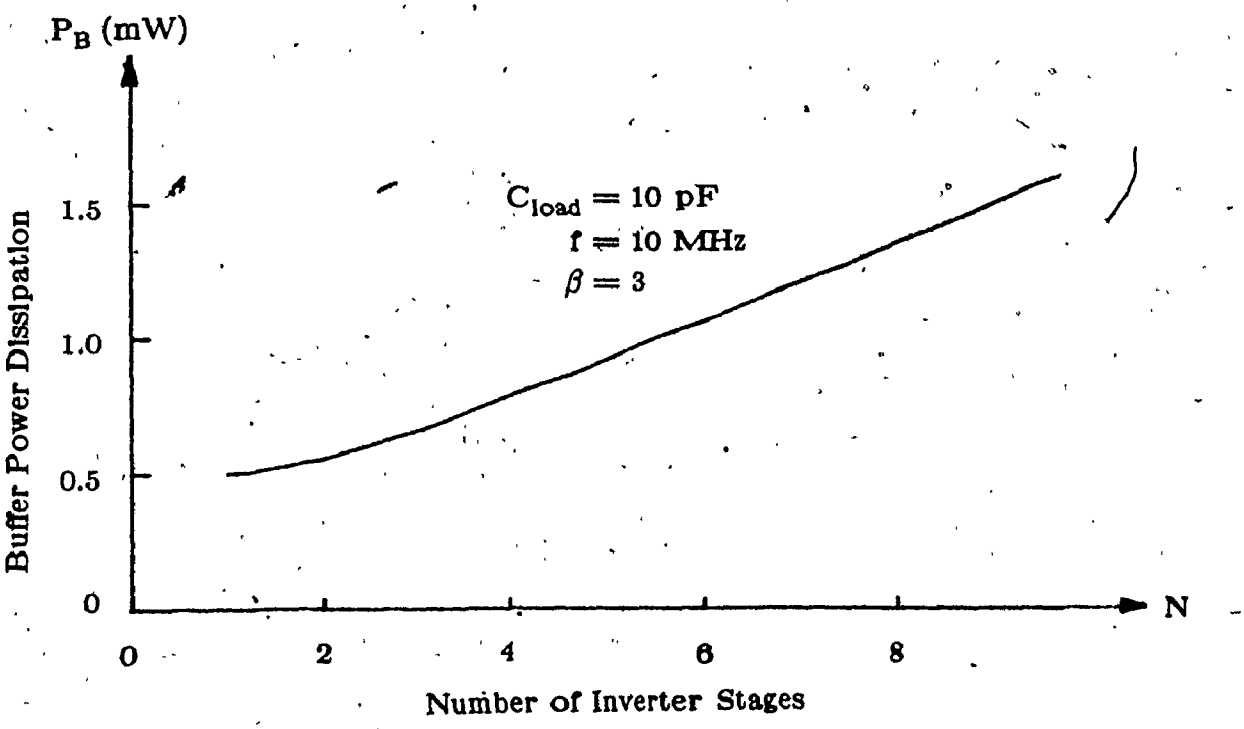


Figure 4.4.2 Power Dissipation as a Function of Number of Inverter Stages

CHAPTER 5

OPTIMIZATION OF CMOS DEVICE

5.1 Introduction

This chapter describes optimization procedures for area, time and power as well as the combinations of these parameters. An analytical expression of area for the single and the multi-stage buffer have been presented in Section 5.2.1 and 5.2.2 respectively. Special attention is focused on the general analytical expression to determine the width of each inverter for the design of multi-stage buffer.

In Section 5.3.1 analysis of various objective functions are presented. Recommendations are made in developing algorithms for our implementation for the design of buffer. These algorithms are used to choose the optimum performance parameters such as scaling factor and number of inverter stages with respect to an objective function such as propagation delay, area, and power dissipation as well as their combinations. In Section 5.3.2, power and delay tradeoff has been described and the feasible tradeoff boundary of power and delay has been found according to Matson's optimum loop [21]. A general description of the combinational products of area, power and delay is described in Section 5.3.3.

5.2 Area of CMOS Device

5.2.1 A Single CMOS Inverter

The area in a single CMOS inverter consists of two components:

1. gate area
2. diffusion area

According to Fig. 2.2.3 , the area of gate for a single CMOS inverter is given by

$$A_g = W_n L_n + W_p L_p$$

$$= W_n L_n (1 + \beta) \tag{5.2.1}$$

and the area of diffusion for a single CMOS inverter is similar as

$$A_d = AD_{dn} + AD_{dp} + AD_{sn} + AD_{sp} \tag{5.2.2}$$

where AD_{dn} and AD_{dp} are the drain diffusion areas, similarly AD_{sn} and AD_{sp} are the source diffusion areas of N- and P-channel transistors respectively. These areas determined by geometrical size from Fig. 2.2.3, for the given design rule. We assume that the length of the drain or source areas to be equal to L_n of N-channel transistor for all regions, then the are a of diffusions A_d can be rewritten as

$$A_d = 2 W_n L_n (1 + \beta) \tag{5.2.3}$$

The area of a single CMOS inverter is the sum of gate and diffusion areas and is represented by

$$A_I = A_g + A_d = 3 W_n L_n (1 + \beta) \tag{5.2.4}$$

i.e. the area of a single CMOS inverter depends on device parameters β , W_n and L_n .

5.2.2 CMOS Buffer

The area of a buffer is the sum of each individual inverter's area, For the finite N-stage CMOS inverter buffer, the buffer area can be written by

$$A_B = \sum_{i=1}^{N} A_{Ii} \tag{5.2.5}$$

where A_{Ii} is the area of i th stage inverter and the relationship between this gate capacitance C_{gi-1} at the node $i-1$ and the load capacitor C_{load} according to Eqn. (3.3.1) for ($i = 1, \ldots, N$) is determined by

$$C_{load} = S^{N-i+1} C_{gi-1} \tag{5.2.6}$$

where C_{gi-1} is determined by

$$C_{gi-1} = W_{ni} L_{ni} C_{ox} (\delta_1 + \delta_2 \beta) \tag{5.2.7}$$

and W_{ni} and L_{ni} are the width and length of N-channel transistor of the *ith* stage inverter respectively. Combining Eqns. (5.2.6) and (5.2.7), W_{ni} is determined by

$$W_{ni} = \frac{C_{load}}{S^{N-i+1} C_{ox} L_{ni} (\delta_1 + \delta_2 \beta)} \tag{5.2.8}$$

This is a general expression to determine the N-channel width of each inverter for the design of each buffer stage except the width of the first inverter has option. For simplicity, we have adopted the minimum size inverter as a first stage of the CMOS buffer in our implementation. As the design process is given, let $L_{ni} = L_n$ for all lengths of N-channel transistor inside the buffer. Substituting Eqns. (3.3.10), (5.2.4) and (5.2.8) into Eqn (5.2.5). The total area of the buffer, A_B can be rewritten as

$$A_B = \frac{(1 - Y) g_4}{(1 - S) Y C_{ox}} C_{load} \tag{5.2.9}$$

since $C_{load} = S^N C_{go}$, we insert it into Eqn. (5.2.9) and then A_B can be

represented as a function of the first stage inverter

$$A_B = \frac{(1 - Y)g_4}{(1 - S)C_{ox}} C_{go} \tag{5.2.10}$$

where

$$g_4 = 3\frac{(1 + \beta)}{(\delta_1 + \delta_2\beta)} \tag{5.2.11}$$

is a design process dependent constant, typical value of g_4 by varying β for the given process parameters is given in Table A-3 of Appendix A. Fig. 5.2.1 shows area of buffer A_B as a function of the scaling factor and Fig. 5.2.2 shows that A_B is a function of the number of inverter stages.

5.3 Objective Functions Optimization

5.3.1 Buffer Area Optimization

So far, we have minimized the buffer propagation delay and power dissipation without looking into the area. Minimization of area is important because the need for large buffers is always there, i.e. the I/O pads, the clock drivers, circuit isolators etc. These buffers usually occupy a relatively large part of the total area of the chip, and secondly the cost of fabricating a circuit is an exponential function of the area. In this case, optimizing the area of a buffer design is much more important than optimizing other functions.

For instance, for a given process parameter and normal working condition ($\beta = 3$, $V_{DD} = 5\ V$, $f = 10\ MHz$), to drive a 15 pF of loading capacitance, the optimum number of inverter stages N_o is 4.77 (optimum scaling factor S_o is 3.37) when delay is minimized. If we choose the integer number of $N = 4$ instead of $N = 5$, then speed is lowered by factor of 2%, but 32.8% of area is

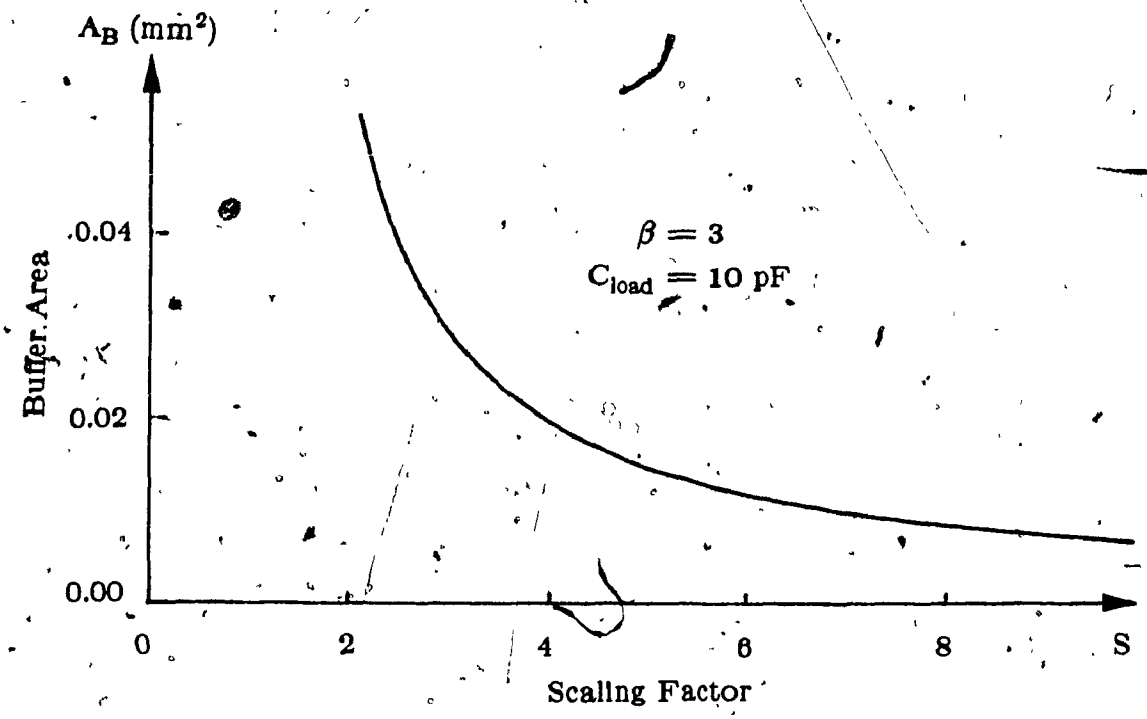


Figure 5.2.1 Buffer Area as a Function of Scaling Factor

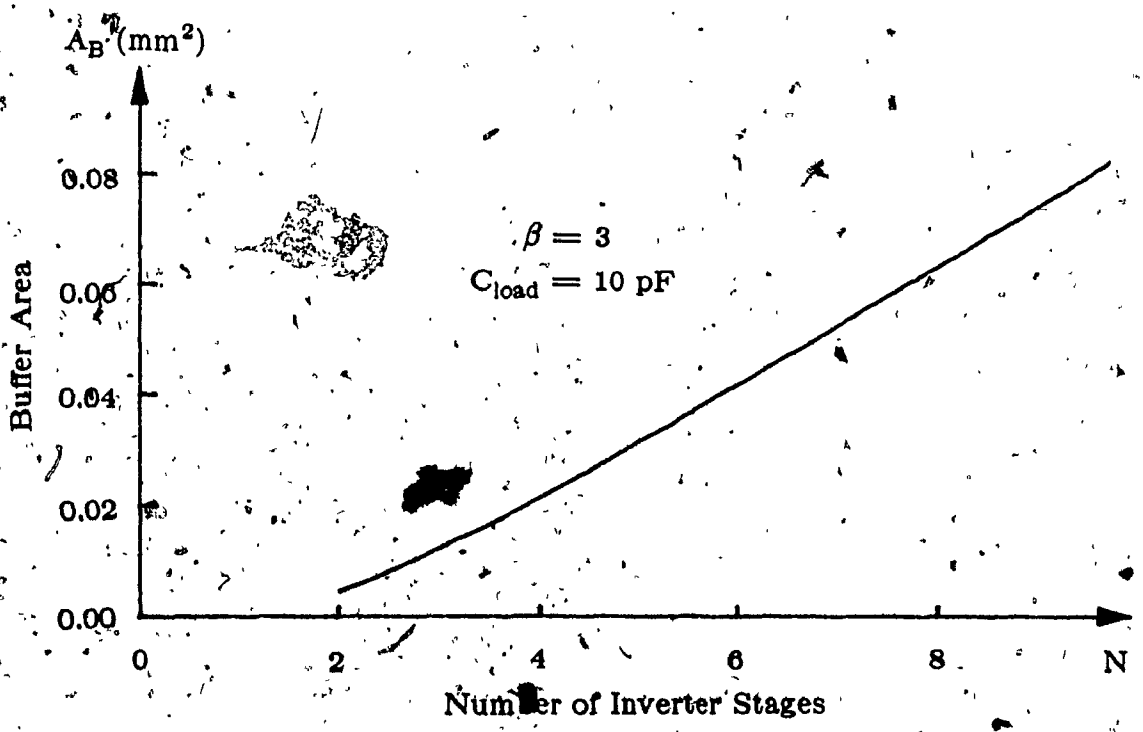


Figure 5.2.2 Buffer Area as a Function of Number of Inverter Stages

saved. The power is a similar case.

Fig. 5.3.1 is a plot of area, propagation delay and power dissipation as a function of scaling factor S. As the scaling factor increases, Power, area, and delay are relatively decreased or increased in corresponding scaling factor region, since the optimum value of the scaling factor for various objective functions differ, then one has to be more observant of the scaling factor that we choose to optimize as many functions as possible.

5.3.2 Power-Delay Tradeoff

Fig. 5.3.1 does not offer a solution when we require to optimize two objective functions as a product. We know however that power and delay can be traded for each other. A classic power-delay tradeoff curve is given by Matson's optimum loop [21]. For the example buffer and for a given process this is shown in Fig. 5.3.2. This shows that the pair set ρ of points in the PD plane could be presented as a function of any performance parameter either $\rho = f(S)$ or $\rho = f(N)$. Therefore the points at the segmental border of the feasible curve of power-delay tradeoff probably can be determined as

$$\rho = \begin{cases} \rho_1^* \\ \rho_2^* \end{cases} \tag{5.3.1}$$

where ρ_1^* is the pair set of border point at the lowest level of propagation delay and the ρ_2^* is the pair set of border point at the lower level of power dissipation. Fig. 5.3.2 describes the power-delay tradeoff boundary variation, as seen the bold line curve moves from ρ_1^* to ρ_2^*, we trade off speed for reduced power dissipation. Our delay specification restricts the points that we can accept to those losing speed less than or equal to D^* which is computed by our power-delay tradeoff algorithm. This algorithm is implemented by taking the derivative $d(t)/d(S)$ all

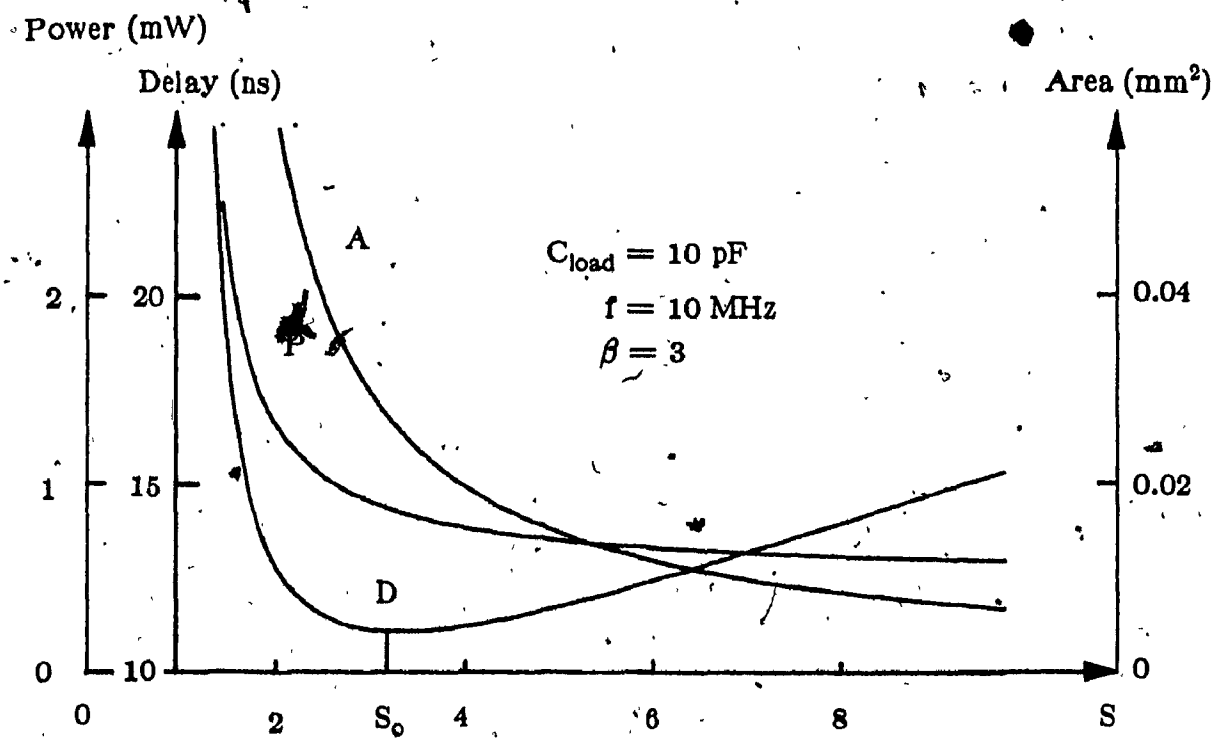


Figure 5.3.1 Area, Power and Delay as a Function of Scaling Factor

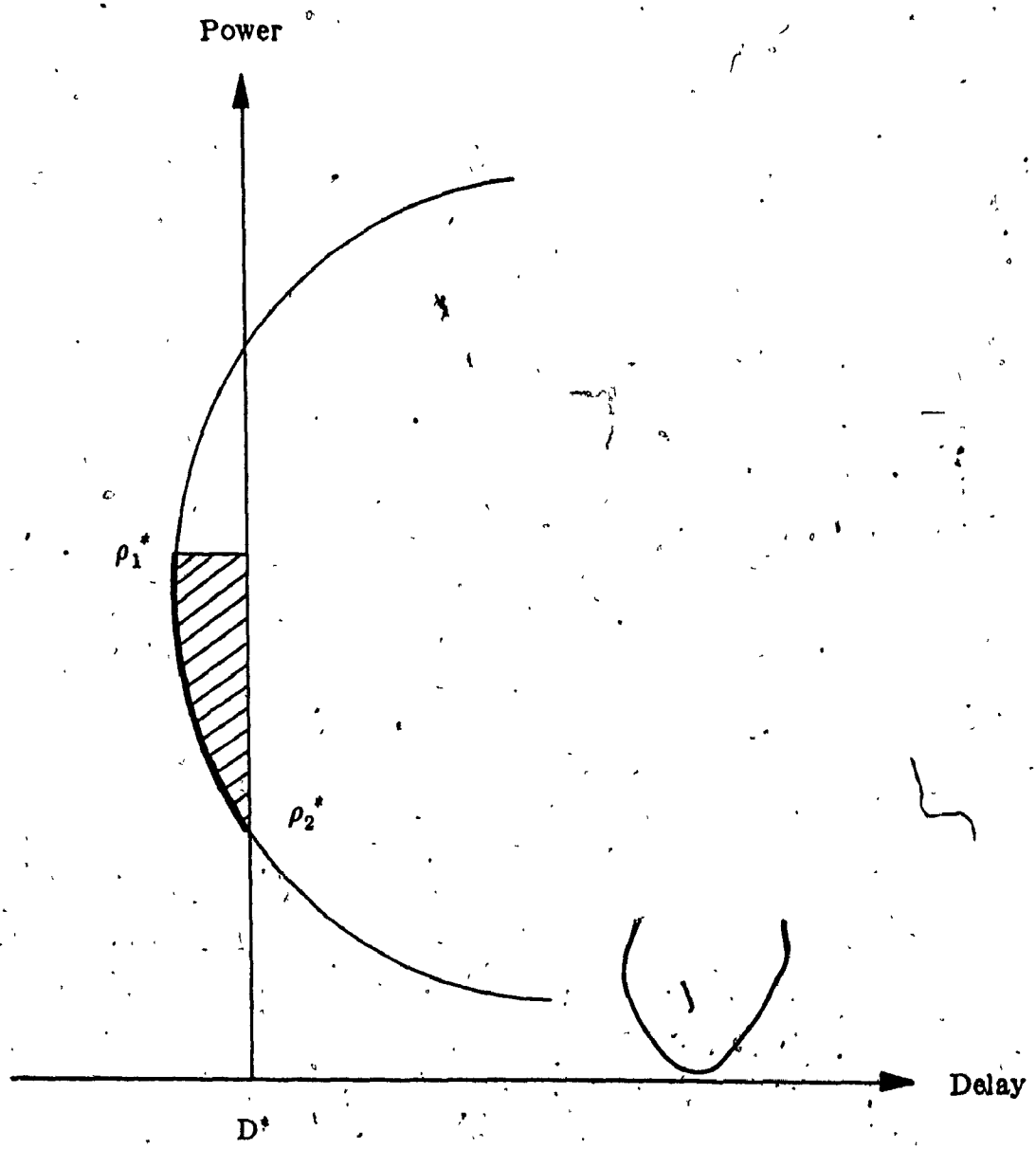


Figure 5.3.2 Variation of Power-Delay Tradeoff Boundary

along the delay curve at ΔS intervals and evaluating the ratio of rate of change of speed with scaling factor at various successive points on the delay-scaling factor curve. The direction of derivative can be mathematically expressed as

$$\begin{cases} d(t)/d(S) < 0 \\ d(t)/d(S) = 0 \\ d(t)/d(S) > 0 \end{cases} \tag{5.3.2}$$

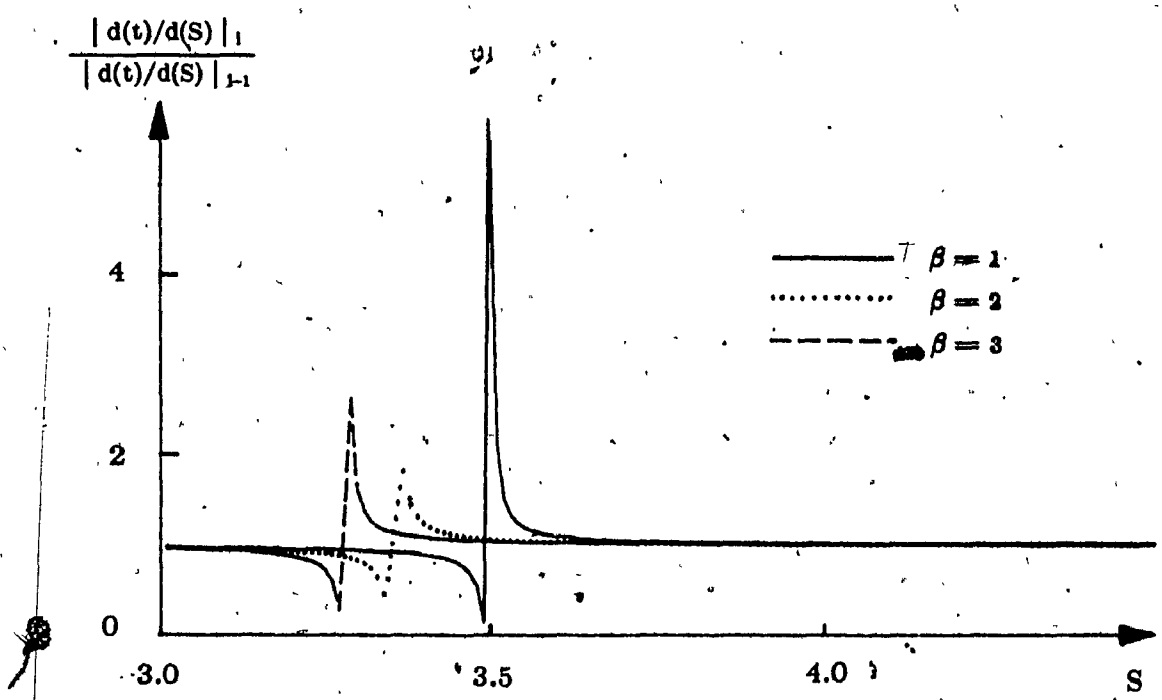
Notice that $d(t)/d(S) < 0$ indicates "delay decrease" and is the region to the left of the minimum delay points; also since the derivative of minimum delay is zero i.e. $d(t)/d(S) = 0$; and $d(t)/d(S) > 0$ shows "delay increase" and is the region to the right of the minimum delay. Our interest here is on the last region.

Fig. 5.3.3 is a plot of the ratio of derivative variation with respect to scaling factor. The ratio of rate of change is great in the region of the minimum delay with the rest of curves being almost constant. This gives the exact point of minimum and the areas of the curve which are suitable for tradeoff. Subscribe $i-1$ and i is the calculation step of S. The following is an example that shows the percentage of speed traded out and optimal power traded in as a function of channel width ratio β, and with $C_{load} = 10\ pF$ and $f = 10\ MHz$ as parameters for a particular process.

β	Speed trades out	Power dissipation deducted
1	9.5%	21.3%
2	10.2%	21.5%
3	10.65%	21.4%

5.3.3 PD, AT and AT^2

We have discussed the optimization of the buffer behaviors when the objective function is propagation delay, area, power dissipation. We are interested in the minimization of some combinations of the area, power and



Scaling Factor

Figure 5.3.3 Ratio of Derivative Variation as a Function of Scaling Factor

propagation time for the CMOS buffer. Frequently, these objective functions are expressed as AT, AT^2 and PD, i.e. the products of area-time, area and square of the time as well as power-delay.

A useful and often-quoted figure of merit for CMOS device is the power-delay product (PD), also the PD could be a most straight forward case of $P^{k_1}D^{k_2}$ while k_1 and k_2 are 1. Where k_1 and k_2 are the dimensions of the power and delay respectively, The units of PD are (watts)(seconds) = joules. Thus PD may be interpreted as energy consumed per logic decision.

Fig. 5.3.4 illustrates objective functions such as AT, AT^2 and PD as a function of scaling factor S for some design parameters of $\beta = 3$, $f = 10\ MHz$, $Cload = 10\ pF$ and $V_{DD} = 5\ V$. As we see AT follows the same curve as the area and thus it has minimum at ∞. But the AT^2 has the minimization as optimum scaling factor $S_0 = 11.8$. Well, as we know, that these results are expressed as AT and AT^2 bounds are required to solve problems in the VLSI domain. In order to solve a certain problem, at least some minimum amount of information must be shipped from one part of the circuit to another, or else one half of the chip can be fooled into thinking the input to the other half was one thing, when in fact it was another. Since shipping large amounts of data requires either a long time or many parallel wires, we can obtain AT or AT^2 lower bounds in this way. These bounds are based on the requirements for information flow within the chip.

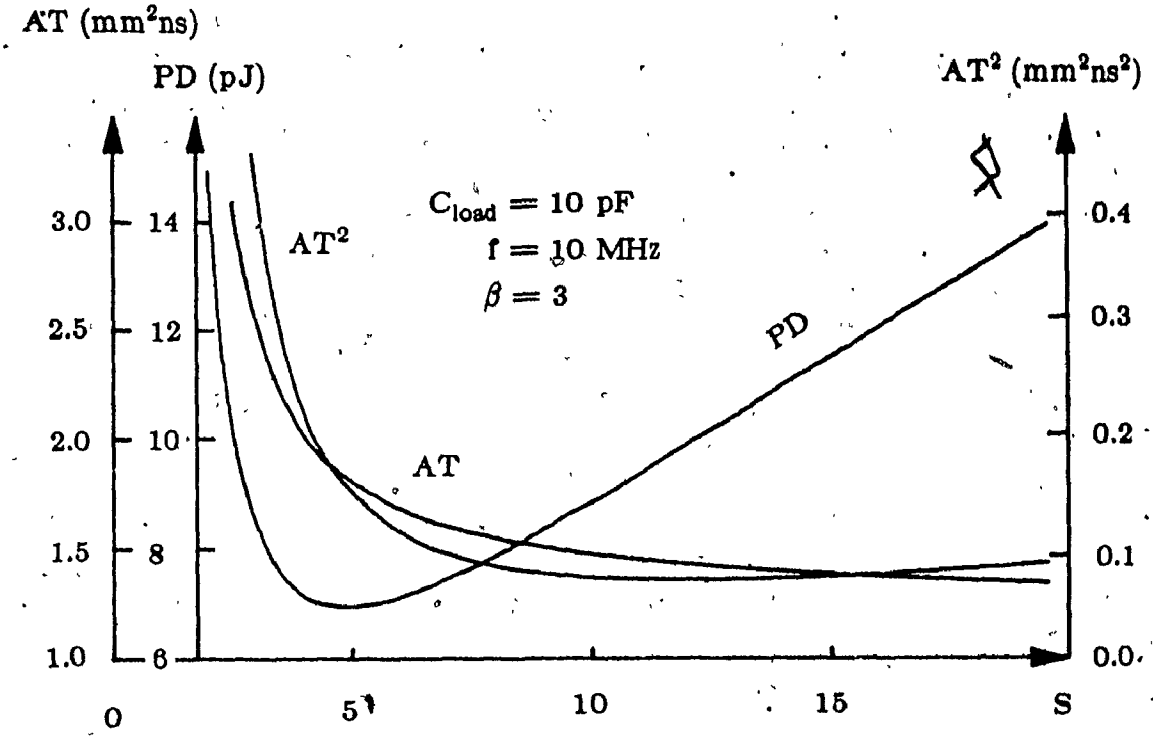


Figure 5.3.4 Variation of Objective Functions with Scaling Factor

CHAPTER 6

IMPLEMENTATION

6.1 Introduction

In order to create a full custom layout of CMOS buffer at mask level, a description language such as Caltech Intermediate Form - CIF file is used. Initially however design specification have to be supplied by the user. These are parameters such as the objective function or the layout type or the loading factor. The module generator uses the user specified data to arrive at its optimized buffer design, through the algorithms that have been described in previous chapters. The implementation of these algorithms are enforced by the module which contains a set of programs written in C. The program contains both the a) optimization algorithms and b) design rule. Hence output is optimized and correct with respect to both the functional specification and the design rules.

This module is composed of four major components, these are a) Performance optimization, b) Design parameters calculation, c) Poly routing configurations classification and d) Layout generation. The flowchart given in Fig. 6.1.1 describes the basic operation of this module. In the following description we will attempt to describe the program operation in conjunction with its implementation and reference to the theoretical results.

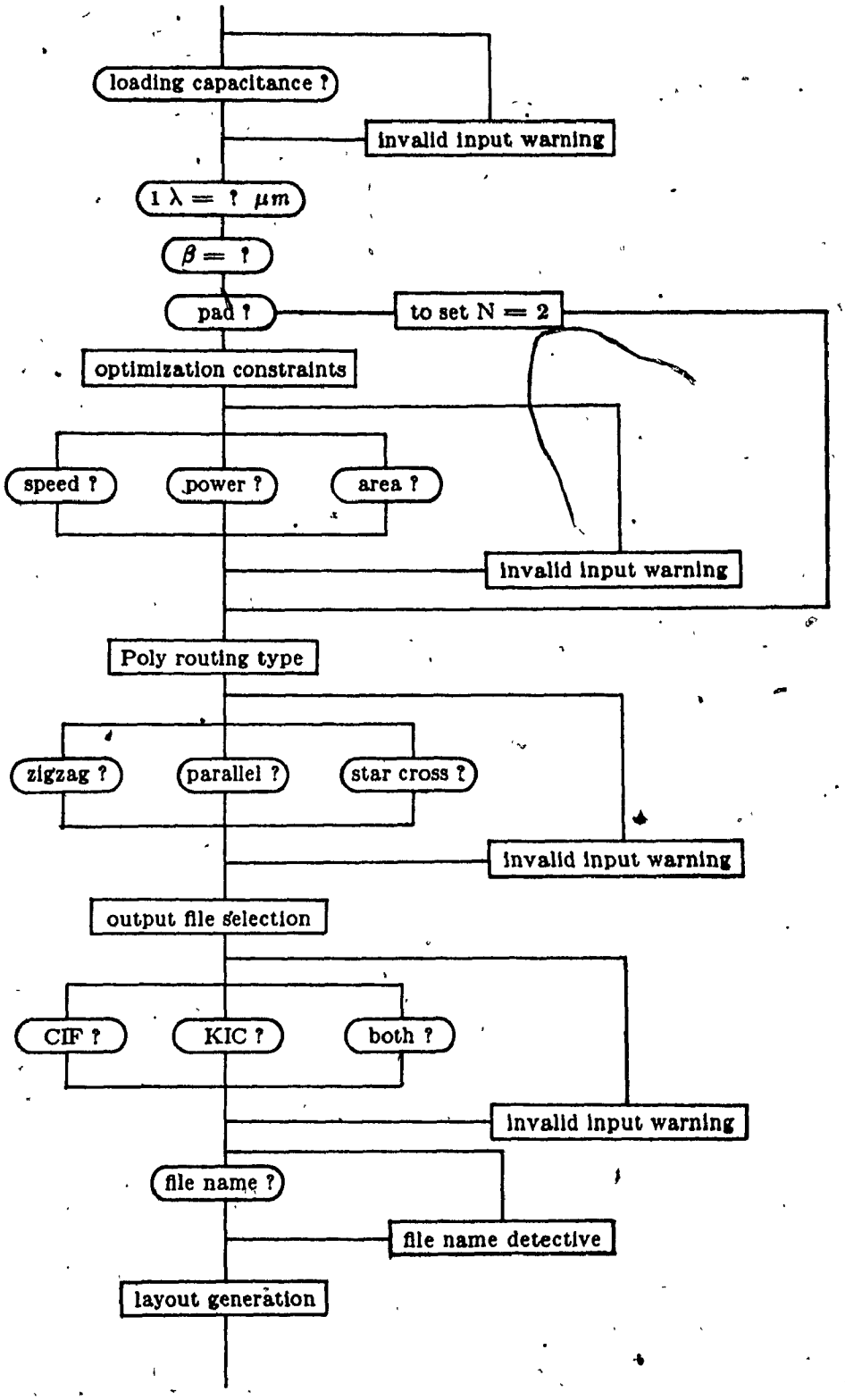


Figure 6.1.1 Diagram of Basic Operation

[1] The user is prompted to decide whether a conventional CMOS buffer or an output pad is required. If an output pad has been chosen, a two-stage inverter buffer is selected. In this case, the number of inverter stages, N, is set to 2, and no further optimization is performed.

[2] Design parameters input:

a. As a second item, the program expects a loading capacitor as an input data. The value typed should be in pF. (no units to be typed). The format is free.

b. The user is then prompted for the microns per lambda, (here 2.5 should be typed for a 5 micron process). It must be however emphasized here that the design rules used in the program are those of CMOS-1B NT and any attempt to use other technologies requires further enchancement of the program to include other technology files.

c. The third design parameter input prompted for is the channel width ratio β.

[3] The program automaticly checks to see if the specified fan-out is above a threshold (in this program the fan-out threshold, $Y = C_{load} / C_{go}$, is set to 5). If it is not then warning is given otherwise the optimization continues.

[4] Optimization performance: The user is prompted to input the objective functions as "speed", "power" and "area". Then the algorithms that are described previously invoked to produce the corresponding optimized performance parameter.

[5] Poly routing type selection: Currently, we provide three types of Poly routing structures, they are in traditional zigzag, parallel and star cross types.

[6] Output file format selection: The final layout mask symbolic description is in CIF format, but an attached KIC format is created as an option, which is suitable for viewing on the graphical screens which run KIC2.

6.2 Performance Optimization

The performance optimizer is a program module which trades off speed, power and area to meet user's requirement and affects the size of the buffer cell only. The buffer is usually a chain of inverters, with each buffer driving a larger succeeding inverter. The relation of the number of inverter stage and the size of each stage is the function of the performance optimizer, which trades off between various objective functions using previously defined algorithm to arrive at its result.

To improve speed, or meet a specified speed, the buffer is designed to give greatest speed increase along with feasible lose of power or area consumption. In this case the number of inverter stages are high, thus contributing to large area and large power consumption. There is a relation between speed and area or power, and scaling factor and number of inverter stages.

The following show the two algorithms for speed and power optimization respectively.

Optimum Speed Algorithm

Set *optimum_stage* = ln (Y) / ln (3.37) ;

Let r = *optimum_stage* ;

Compute delay with r and $r+1$ stages;

expression_1 = r ($Y^{1/r}$) + $g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3$) ;

expression_2 = (r + 1) ($Y^{1/(r+1)}$ + $g_1\gamma_1 + g_2\gamma_2 + g_3\gamma_3$) ;

Select N for minimum delay.

Optimum Power Algorithm

Do loop scaling = scaling + increment ;

to compute derivative of delay and

compare the differences of the ratio

of derivative to meet tradeoff.

Let *optimum_stage* = ln (Y) / ln (*scaling*) ;

Set r = *optimum_stage* ;

Set $N = r$.

Optimum area is a similar algorithm to power.

6.3 Design Parameters Calculation

In the above section we gave an algorithm to obtain the optimum number of inverter stages N. Next we recomputer scaling factor S with N as an integer. For instance, if a 15 pF capacitance loading is needed to be driven and we let $\beta = 3$. Then let speed be the optimum function accordingly optimum scaling factor $S_o = 3.37$. This value is then used to determine the optimum number of inverter stages $N_o = ln(Y)/ln(S_o) = 4.77$, but the N_o must be a integer number of r, as a result $r = 5$ is chosen. Now S is needed to be readjusted by $S = Y^{1/N}$. For further calculation we applying equation (5.2.8) ,

$$W_{ni} = \frac{C_{load}}{S^{N-i+1} C_{ox} L_{ni} (\delta_1 + \delta_2 \beta)}$$

which gives the values of N-channel width of each inverter as follows

Stage's number	1	2	3	4	5
W_n (μm)	5	16	49	152	474

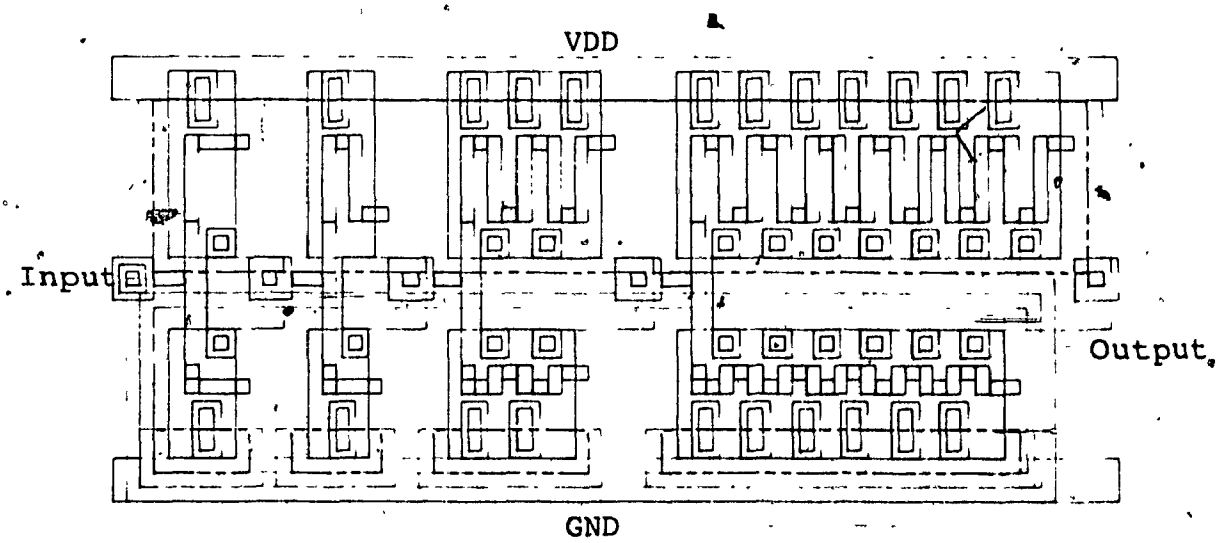
6.4 Physical Layout Consideration

Basically, we have three type of layout configurations. Poly routing in

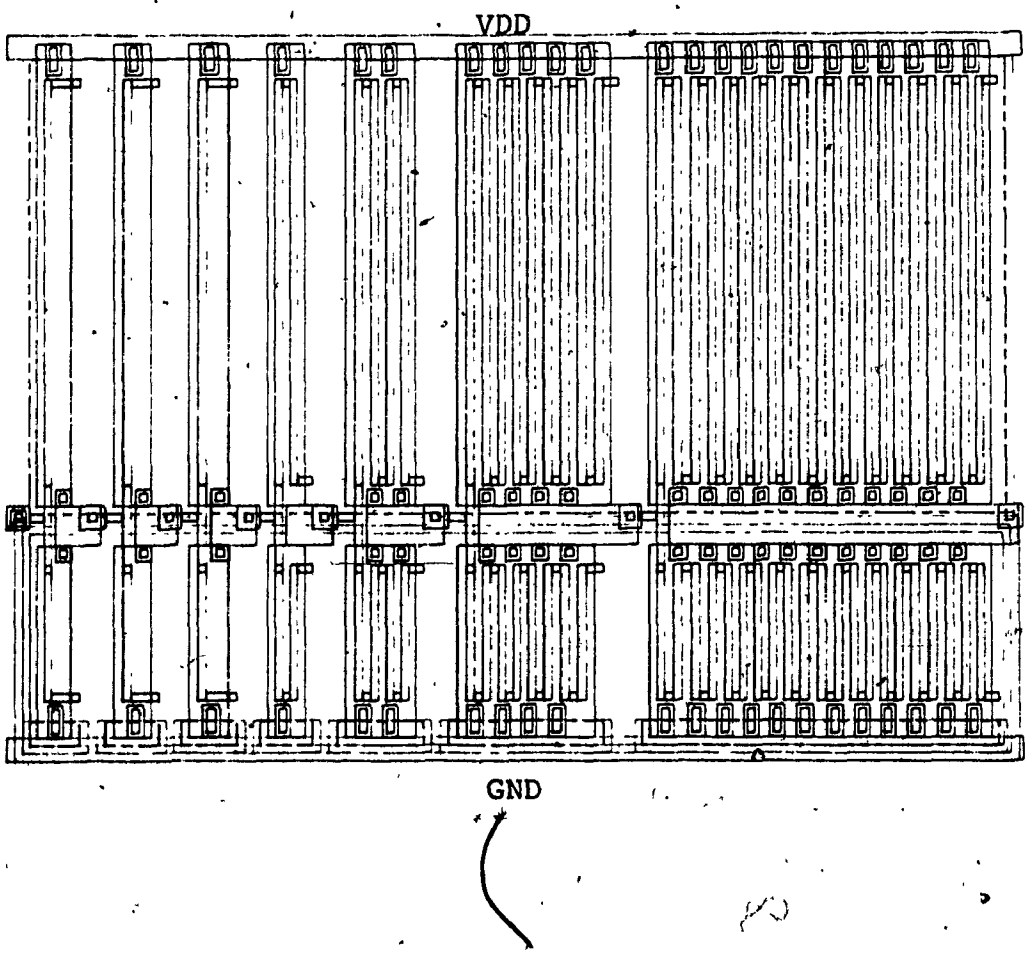
1. Zigzag
2. Parallel
3. Star cross

Running polysilicon in zigzag is a traditional technique for long polysilicon running. Alternatively, a parallel connection style may be used to strap polysilicon to reduce delays due to long poly runs particularly for the big value of channel width, the source and drain regions are stitched with the contacts and metal to reduce source-drain resistance. A further reduction in drain capacitance is achieved by using the star cross connection. In mask, the source and the drain regions would be one continuous area with no corner gaps to increase gain and reduce peripheral capacitance (due to the drain diffusion area shares with contact). Fig. 6.3.1 (a) and (b) illustrate the symbolic layout in color. (a) represents a 4-stage inverter buffer in zigzag configuration with respect to $\beta = 3$, (b) is a 7-stage inverter buffer with large capacitive load to drive. The parallel configuration buffer is represented in Fig. 6.3.2 with respect to $\beta = 3$ and 7-stage inverter. Fig. 6.3.3 shows the star cross configuration buffer with respect to $\beta = 1$ and 1-stage inverter.

The colors of the plot are corresponding as following:



(a) A 4-stage Inverter Buffer Layout



(b) A 7-stage Inverter Buffer Layout

Figure 6.3.1 CMOS Buffers in Zigzag Configuration

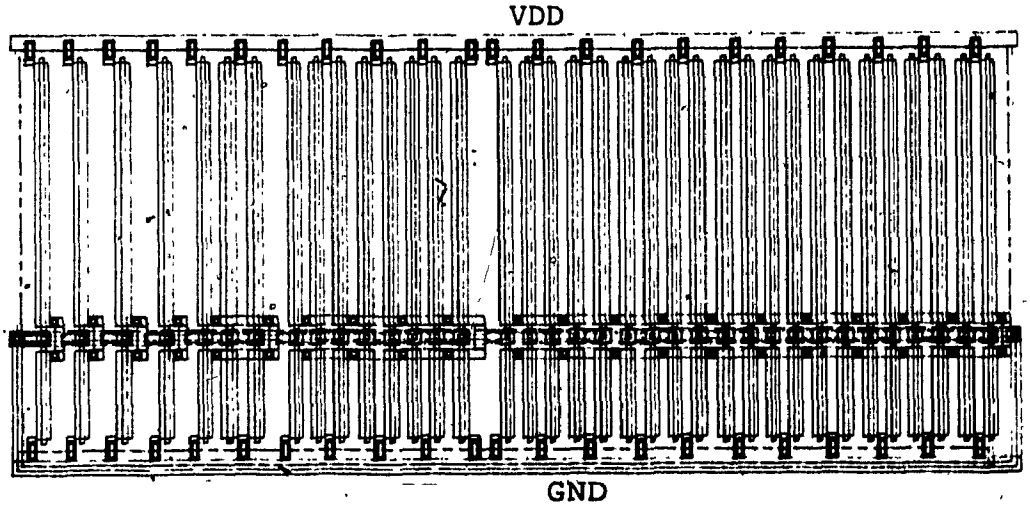


Figure 6.3.2 A 7-stage CMOS Inverter Buffer Layout

In Parallel Configuration

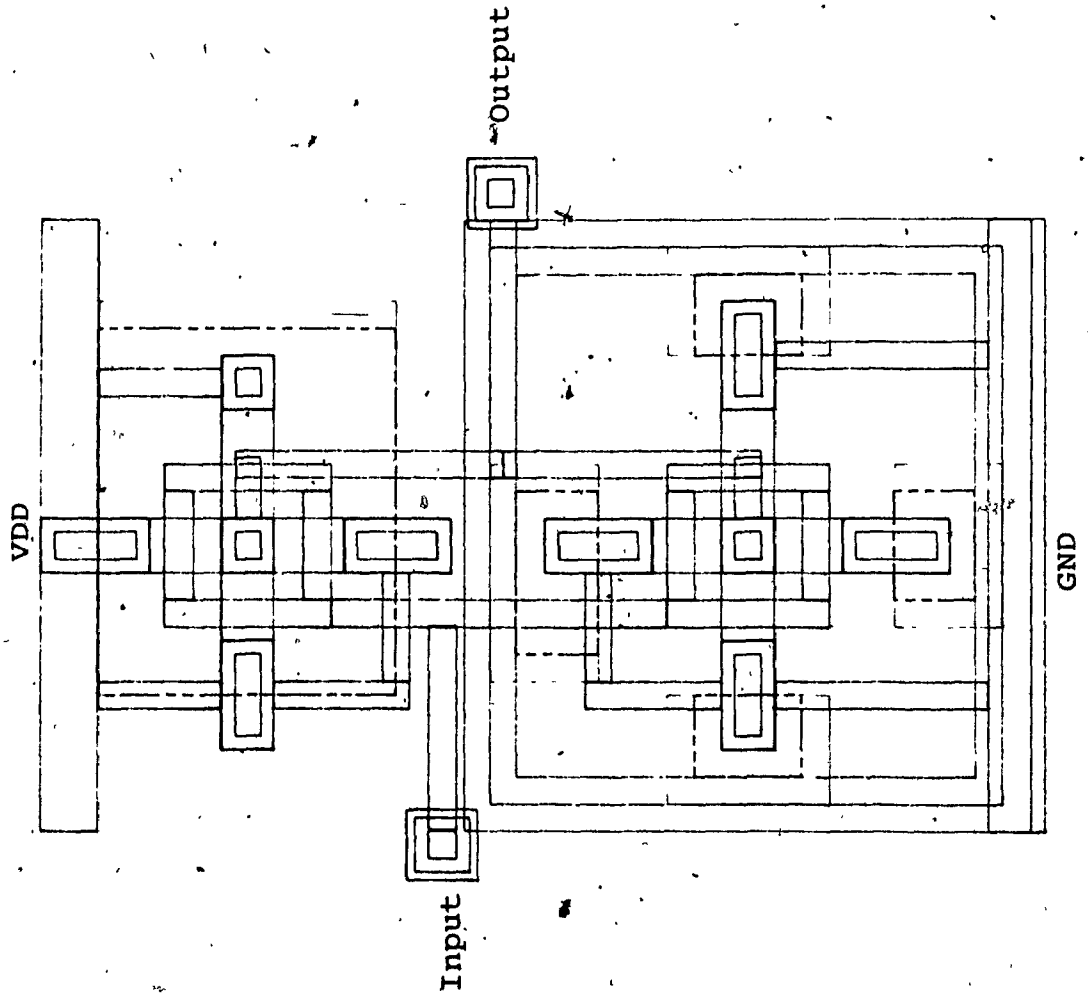


Figure 6.3.3 A Single Stage CMOS Inverter Buffer

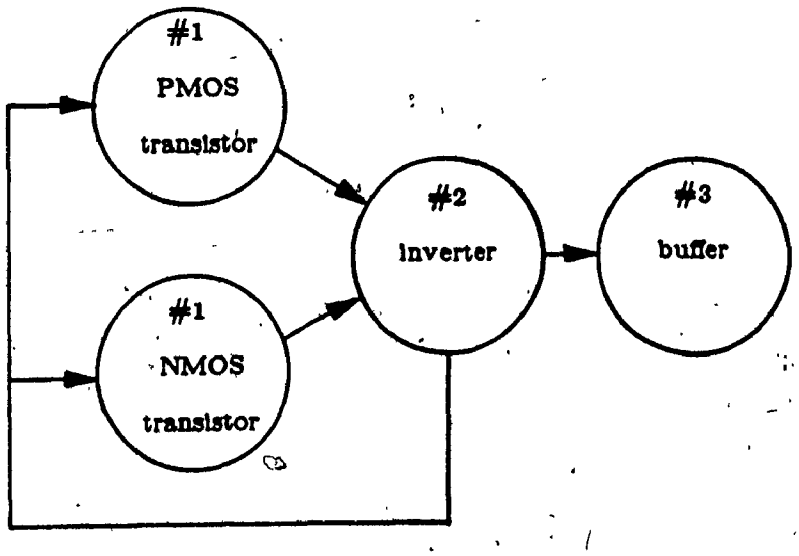
Layout in Star Cross Configuration

Layer	Plotting color
P-well	Brown
Device well	Blue
P-guard	Yellow
N-guard	Green
Polysilicon	Orange
N+ Doping	Red
P+ Doping	Dotted-Blue
Contact	Green
Metal	Black

6.5 Layout Generation

A Lambda-based CMOS-1B process layout rules is used for our layout generator. The process and rules description are identical to those contained in CMC publication GICIS version 3:0 issued on January 1987 [33].

The layout generation is done by a set of software modules. The sequence is shown as following:



This sequence shows that a primary module is used to create the PMOS and NMOS transistors for different configurations as required. A secondary module is used to assemble place and rout the strip PMOS and NMOS transistors to form an inverter and the modules are called repeatedly depending on the number of inverter stages in the buffer. The last module assembles the buffer by place and rout the strip inverters to create the final buffers.

As an example, we illustrate the algorithm used to create a parallel Poly routing.

Parallel Poly Routing Algorithm

P-channel transistor;

Step 1: Calculate the averaged height of Poly in integer form calculated as

height = (int) (100 * total_width) / n ;

Step 2: Set threshold height for P-channel transistor calculated as

threshold = a * beta * height;

Step 3: Set the minimum height of Poly using given design rules;

Step 4: Calculate the number of vertical Polys using design rules and step 3;

Step 5: Calculate Poly positions starting with initial placement.

The other configurations are similar to the above algorithm and the detailed programs are given in appendix D.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This thesis contained two parts:

1. CMOS performance analysis
2. Implementation of the module generator for buffers.

In part 1, we discussed in detailed the electrical behavior associated with CMOS inverter and the modeling of various parameters, such as capacitance, delay to a ramp input signal, power dissipation and area consideration. These models were formalized in an analytical expressions taking the layout parameters into account. Using these expressions we developed optimum expressions for the scaling factor and the number of inverter stages to various objective functions. These expressions have been represented in terms of process constants, so that it is very flexible to apply it to processes.

In part 2, the buffer module generator is implemented by C. The module performs CMOS buffer automated design, performance optimization and layout generation in multiple configurations under the UNIX bsd 4.3 operating system on Vax and Sun workstations. The module has the following features:

1. It generates layouts in multiple configuration as specified by the user.
2. It optimizes buffer design in terms of the multiple objective function as specified by the user.
3. Process and layout parameters are easily updatable through technology file.

The module was written for CMOS-1B *NT* process. If is to be useful and adaptive in the future, it is essential to extend the work in such a way as to make it technology independent.

REFERENCES

[1] David A. Hodges, Horace G. Jackson, *Analysis and Design of Digital Intergrated Circuits*, McGraw-Hill, 1983.

[2] Neil H. E. Weste, Kamran Eshraghian, *Principles of CMOS VLSI Design, A Systems Perspectine*, Addison-Wesley, 1985.

[3] Amar Mukherjee, *Introduction to nMOS & CMOS VLSI System Design*, Prentice-Hall, 1986.

[4] Donald E. Ward, Robert W. Dutton, "A charge-oriented model for MOS transistor capacitances," *IEEE Journal of solid-state circuits.* vol. sc-13, no. 5, October 1987, P.P. 703-707.

[5] Norman P. Jouppi "Timing analysis for nMOS VLSI," *IEEE 20th Design Automation Conference*, 1983, pp. 411-418.

[6] J. R. Burns, "Switching response of complementary-symmetry MOS transistor logic circuits," *RCA Rew.*, vol. 25, pp.627-661, Dec. 1964.

[7] Mark D. Matson, "Macromodeling of digital MOS VLSI circuits," *Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology*, VLSI Memo no. 84-212, Nov. 84.

[8] J. Ousterhout. "A switch-level timing verifier for digital MOS VLSI," *IEEE Trans. Computer-Aided Design*, vol, CAD-4, pp. 336-349, 1986.

[9] T. Tokuda, K. Okazaki, K. Sakashita, I. Ohkura, and T. Enomoto, "Delay-time modeling for ED MOS logic LSI," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp.129-134, July 1983.

[10] R. J. Bayruns, R. L. Johnston, D. L. Fraser, Jr., and S. C. Fang, "delay analysis of Si NMOS-G bit/s logic circuits," *IEEE J. Solid-State Circuit*, vol.

sc-19, pp.755-764, Oct. 1984.

[11] Nils Hedenstierna and Kjell O. Jeppson, "CMOS circuit speed and buffer optimization," *IEEE Trans. Computer-Aided Design*, Vol. CAD-6, No.2, pp.270-281, March 1987.

[12] A. M. Mohsen and C. A. Mead, "Delay-time optimization for driving and sensing of signals on high-capacitance paths of VLSI systems," *IEEE J. Solid-State Circuit*, April 1979.

[13] E. Seewarm, "Switching speeds of MOS inverters," *IEEE J. Solid-State Circuits*, vol. sc-15, pp.246-252, Apr. 1980.

[14] A. Kanuma, "CMOS circuit optimization," *Solid-State Electron*, vol. 26, pp. 47-58, 1981.

[15] Mead, C., and L. Conway (Eds.), *Introduction to VLSI System*, Reading, Mass.: Addison-Wesley, 1980.

[16] M. I. Elmasry, "Digital MOS intergrated circuits: A tutorial," *Digital MOS Integrated Circuits*. New York: *IEEE Press* pp. 4-27, 1981.

[17] Harry J. M. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *IEEE J. Solid-State Circuit*, vol. sc-19, no.4, pp. 468-473, Aug. 1984.

[18] Stephen Trimberger, "Automated performance optimization of custom integrated circuits", *Computer-Aided Engineering Design*, vol. 1, pp. 525-283, 1985.

[19] Lance A. Glasser and Lennox P. J. Hoyte, "Delay and power optimization in VLSI circuits," *Proceedings of the 21st Design Automation Conference*, pp. 529-535, 1984.

[20] C. Seitz, A.H.Frey, S. Mattison, S. D. Rabin, D. A. Seck, and J. L. Van de Snepscheut, "Hot-clock nMOS," *Proc. 1985 Chapel Hill Conference on Very Large Scale Integration, (Ed. H. Fuchs), Chapel Hill, NC.: Computer Science Press*, pp. 1-17, 1985.

[21] Mark D. Matson "Optimization of digital MOS VLSI circuits"

[22] Jeffrey D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, 1984.

[23] Ronald F. Ayres, *VLSI silicon Compilation and the Art of Automatic Microchip Design*, Prentice-Hall, 1983.

[24] B. Ackland and N. West, "An Automatic Assembly Tool for Virtual Grid Symbolic Layout," *Proc. VLSI'83*, North Holland Publishing, August 1983, Trondheim, pp. 457-466.

[25] M. R. Buric, C. Christensen and T. G. Matheson, "PLEX: Automatically generated microcomputer layouts," *Proc. IEEE International Conf. on Computer Design (ICCD'83)*, Port Chester, NY, October 1983, pp. 181-184.

[26] C. M. Lee, B. R. Chawla and S. Just, "Automatic generation and characterization of CMOS polycells," *Proceeding of the 18th Design Automation Conference*, June 1981, pp. 220-224.

[27] G. Persky, D. N. Deutsh and D. G. Schweikent, "LTX - A minicomputer-based system for automated LSI layout," *Journal of Design Automation and Fault Tolerent Computing*, vol. 1, no. 3, May 1977, pp. 217-255.

[28] O. Wing, "Automated gate-matrix layout," *Proc. IEEE International Symposium on Circuits and Systems*, 1982, pp. 681-685.

[29] A. E. Dunlop, "Automatic layout of Gate Arrays," *Proc. of the IEEE Symposium on Circuit and Systems*, 1983, pp. 1245-1248.

[30] David E. Krekelberg, Eugene Shragowitz, Gerad E. Sobelman, and Li-Shin Lin, "Automatic layout synthesis in the YASC silicon compiler," *23rd ACM/IEEE Design Automation Conference*, 1986, pp. 447-453.

[31] Shigeo Noda, Hitoshi Yoshizawa, Etsuko Fukuda, Haruo Kato, Hiroshi Kawanish and Takashi Fujii, "Automatic layout algorithms for function blocks of CMOS gate arrays," *22rd ACM/IEEE Design Automation Conference*, June 1985, pp. 46-52.

[32] A. Vladimirescu, Kaihe Zhang, A. R. Newton, D. O. Pederson, A. Sanglovanni- Vincentelli, "SPICE user's guide" version 2G6, *Department of Electrical Engineering and Computer Sciences, University of California, Berkeley,*

[33] CMC, "Guide to the integrated circuit implementation services of the canadian microelectronics corporation," *Carruthers Hall, Queen's University,* GICIS version 3:0, January 1987.

APPENDIX A

Table A-1

SPICE Version 2G6 Northern Telecom CMOS-1B Transistor Model Parameters

Symbol	Name	Parameter	Unit	P-channel	N-channel
	LEVEL	Model index		2	2
V_{TO}	VTO	Zero-bias threshold voltage	V	-0.900	0.900
k'	KP	Transconductance parameter	A/V^2	9.75E-06	3.05E-05
γ	GAMMA	Bulk threshold parameter	$V^{1/2}$	0.634	1.592
$2\mid\Phi_F\mid$	PHI	Surface potential	V	0.612	0.695
λ	LAMBDA	Channel-length modulation	$1/V$	3.00E-02	1.00E-02
r_d	RD	Drain ohmic resistance	Ω	2.00E+00	2.00E+00
r_s	RS	Source ohmic resistance	Ω	2.00E+00	2.00E+00
C_{bd}	CBD	Zero-bias B-D junction capacitance	F	2.00E-14	2.00E-14
C_{bs}	CBS	Zero-bias B-S junction capacitance	F	2.00E-14	2.00E-14
I_s	IS	Bulk junction saturation current	A	1.00E-14	1.00E-14
Φ_s	PB	Bulk junction potential	V	0.700	0.700
	CGSO	Gate-source overlap capacitance per meter channel width	F/m	2.44E-10	2.84E-10
	CGDO	Gate-drain overlap capacitance per meter channel width	F/m	2.44E-10	2.84E-10
	CGBO	Gate-bulk overlap capacitance per meter channel length	F/m	2.00E-12	2.00E-12
	RSH	Drain and source diffusion sheet resistance	Ω	7.50E+01	1.50E+01
C_{j_s}	CJ	Zero-bias bulk junction bottom cap. per square meter of junction area	F/m^2	1.54E-04	3.41E-04
M	MJ	Bulk junction bottom grading coef.		0.500	0.500
	CJSW	Zero-bias bulk junction sidewall cap. per meter of junction perimeter	F/m	4.37E-10	1.09E-09
M	MJSW	Bulk junction sidewall grading coef.		0.500	0.500
	JS	Bulk junction saturation current per square meter of junction area	A/m^2	4.19E-10	1.37E-05
t_{ox}	TOX	Oxide thickness	m	8.50E-08	8.50E-08
N_A or N_D	NSUB	Subtrate doping	$1/cm^3$	1.98E+15	9.92E+15
	TPG	Surface state density	$1/cm^2$	1.000	1.000
X_j	XJ	Metallurgical junction depth	m	9.00E-07	1.00E-06
LD	LD	Lateral diffusion	m	6.00E-07	7.00E-07
μ	UO	Surface mobility	$cm^2/V-s$	2.40E+02	7.50E+02
	UCRIT	Critical field for mobility degradation	V/cm	6.44E+04	1.23E+05
	UEXP	Critical field exponent in mobility degradation		0.139	0.022
	VMAX	Maximum drift velocity of carriers	m/s	7.33E+04	4.92E+05
	XQC	Thin-oxide capacitance model flag and coefficient of channel charge share attributed to drain		0.400	0.400

Table A-2

β Independent Process Constants

δ_1	δ_2	δ_3	δ_4	δ_5	γ_1	γ_2	γ_3	γ_4
1.280	1.240	0.448	0.401	1.000	0.441	0.558	0.558	0.113

Table A-3

β Dependent Process-Constants

β	g_1	g_2	g_3	g_4	a	b
1	0.574	0.556	0.556	2.381	0.132	0.098
2	0.504	0.479	0.373	2.394	0.116	0.086
3	0.469	0.441	0.280	2.400	0.111	0.082

Table A-4

Unit delay τ_o by varying β

β	τ_o (ns.)
1	0.384
2	0.337
3	0.322

Table A-5

Input & Output Capacitances of a Single CMOS Inverter by Varying β

β	C_{in} (pF)	C_{out} (pF)
1.0	0.0256	0.0224
1.1	0.0268	0.0228
1.2	0.0281	0.0232
1.3	0.0294	0.0236
1.4	0.0306	0.0241
1.5	0.0319	0.0245
1.6	0.0331	0.0249
1.7	0.0344	0.0253
1.8	0.0356	0.0258
1.9	0.0369	0.0262
2.0	0.0382	0.0266
2.1	0.0394	0.0271
2.2	0.0407	0.0275
2.3	0.0419	0.0279
2.4	0.0432	0.0283
2.5	0.0445	0.0288
2.6	0.0457	0.0292
2.7	0.0470	0.0296
2.8	0.0482	0.0300
2.9	0.0495	0.0305
3.0	0.0507	0.0309

APPENDIX B

The gate oxide capacitance C_{ox} per unit area for both of N- and P-channel transistors is defined by

$$C_{ox} = \frac{\epsilon_{ox}}{t_{ox}} \tag{B-1}$$

where $\epsilon_{ox} = 3.5e-11\ F/m$ and $t_{ox} = 8.5e-8\ m$ are the permittivity and thickness of the gate dielectric and $C_{ox} = 4.06e-4\ F/m^2$.

The device transconductance parameter K_N for N-channel transistor is defined by

$$K_N = \frac{W_n}{L_n} C_{ox}\, \mu_n \tag{B-2}$$

For a $5\mu m$ mini-size inverter $W_n = L_n = 5\ \mu m$, surface mobility of N-channel transistor $\mu_n = 750\ cm^2/V\text{-}s$ and typical value of $K_N = 3.045e-5\ A/V^2$

$$K_{eq} = \frac{-2\Phi_o^M}{V_2 - V_1}\left[(\Phi_o - V_2)^M - (\Phi_o - V_1)^M\right] \tag{B-3}$$

Note that voltage applied to the junction is $V_2 = -(V_{OL} - V_{BB})$ in the low state and $V_1 = -(V_{OH} - V_{BB})$ in the high state. V_{BB} is the (zero or negative) *bodybias* voltage applied to the body with respect to the source of inverter transistors. M is the grading coefficient of bulk junction bottom and sidewall. By convention, voltages applied to junctions are defined as positive for forward bias and negative for reverse bias. On the assumption that $V_{OL} = 0$, $V_{OH} = V_{DD}$, $\Phi_o = 0.7\ V$, $M = 1/2$, $V_{BB} = 0\ V$, this gives $K_{eq} = 0.52$.

APPENDIX C

SIPCE Programs

Input Loading Affects Propagation Delay (Fig. 2.3.4)

```
.model enp pmos(level=2 vto=-0.900 kp=9.75e-06 gamma=0.634 phi=0.612
+            lambda=3.00e-02 rd=2.00e+00 rs=2.00e+00 cbd=2.00e-14
+            cbs=2.00e-14 is=1.00e-14 pb=0.700 cgso=2.44e-10
+            cgdo=2.44e-10 cgbo=2.00e-12 rsh=75.000 cj=1.54e-04
+            mj=0.500 cjsw=4.37e-10 mjsw=0.500 js=4.19e-10 tox=8.50e-08
+            nsub=1.98e+15 tpg=1.000 xj=9.00e-07 ld=6.00e-07 uo=240.000
+            ucrit=6.44e+04 uexp=0.139 vmax=7.33e+04 xqc=0.400)
.model enn nmos(level=2 vto=0.900 kp=3.05e-05 gamma=1.592 phi=0.695
+            lambda=1.00e-02 rd=2.00e+00 rs=2.00e+00 cbd=2.00e-14
+            cbs=2.00e-14 is=1.00e-14 pb=0.700 cgso=2.84e-10
+            cgdo=2.84e-10 cgbo=2.00e-12 rsh=15.000 cj=3.44e-04
+            mj=0.500 cjsw=1.09e-09 mjsw=0.500 js=1.37e-05 tox=8.50e-08
+            nsub=9.92e+15 tpg=1.000 xj=1.00e-06 ld=7.00e-07 uo=750.000
+            ucrit=1.23e+05 uexp=0.022 vmax=4.92e+05 xqc=0.400)
* invert input 2, output 3, vdd 1
.subckt inv 2 3 1
m1u 3 2 1 1 enp w=15u l=5u ad=75p as=75p pd=40u ps=40u
m1p 3 2 0 0 enn w=5u l=5u ad=25p as=25p pd=20u ps=20u
.ends inv
* input 2, output1 3, output2 4, vdd 1
.subckt load1 2 3 4 1
x0 2 3 1 inv
x1 3 4 1 inv
x2 4 5 1 inv
cload1 5 0 0.1pf
.ends load1
* input 1, output1 3, outout2 4, ydd 1, for all follows
.subckt load2 2 3 4 1
x0 2 3 1 inv
x1 3 4 1 inv
x2 3 5 1 inv
x3 4 6 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
.ends load2
.subckt load3 2 3 4 1
x0 2 3 1 inv
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 4 7 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
```

```
cload3 7 0 0.1pf
.ends load3
.subckt load4 2 3 4 1
x0 2 3 1 inv
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 4 8 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
.ends load4
.subckt load5 2 3 4 1
x0 2 3 1 inv
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 4 9 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
.ends load5
.subckt load6 2 3 4 1
x0 2 3 1 inv
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 3 9 1 inv
x7 4 10 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
cload6 10 0 0.1pf
.ends load6
.subckt load7 2 3 4 1
x0 2 3 1 inv
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 3 9 1 inv
x7 3 10 1 inv
x8 4 11 1 inv
```

```
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
cload6 10 0 0.1pf
cload7 11 0 0.1pf
.ends load7
.subckt load8 2 3 4 1
x0 2 3 1 inv
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 3 9 1 inv
x7 3 10 1 inv
x8 3 11 1 inv
x9 4 12 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
cload6 10 0 0.1pf
cload7 11 0 0.1pf
cload8 12 0 0.1pf
.ends load8
.subckt load9 2 3 4 1
x0 2 3 1 inv
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 3 9 1 inv
x7 3 10 1 inv
x8 3 11 1 inv
x9 3 12 1 inv
x10 4 13 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
cload6 10 0 0.1pf
cload7 11 0 0.1pf
cload8 12 0 0.1pf
cload9 13 0 0.1pf
.ends load9
.subckt load10 2 3 4 1
x0 2 3 1 inv
x1 3 4 1 inv
x2 3 5 1 inv
```

```
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 3 9 1 inv
x7 3 10 1 inv
x8 3 11 1 inv
x9 3 12 1 inv
x10 3 13 1 inv
x11 4 14 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
cload6 10 0 0.1pf
cload7 11 0 0.1pf
cload8 12 0 0.1pf
cload9 13 0 0.1pf
cload10 14 0 0.1pf
.ends load10
*  call subcircuits
x0 2 3 4 1 load1
x1 2 5 6 1 load2
x3 2 7 8 1 load3
x4 2 9 10 1 load4
x5 2 11 12 1 load5
x6 2 13 14 1 load6
x7 2 15 16 1 load7
x8 2 17 18 1 load8
x9 2 19 20 1 load9
x10 2 21 22 1 load10
vdd 1 0 dc 5v
vin 2 0 pulse (0 5 0ns 0ns 0ns 20ns 40ns)
.options abstol=1.e-12 vntol=1.e-12 limpts=3000
.tran 0.01ns 29ns
.plot tran v(2) v(3) v(4) v(5) v(6) v(7) v(8) (0,5)
.plot tran  v(2) v(9) v(10) v(11) v(12) v(13) v(14) (0,5)
.plot tran  v(2) v(15) v(16) v(17) v(18) v(19) v(20) (0,5)
.end
```

Output Loading Affects Porpagation Delay (Fig. 2.3.6)

```
.model enp pmos(level=2 vto=-0.900 kp=9.75e-06 gamma=0.634 phi=0.612
+               lambda=3.00e-02 rd=2.00e+00 rs=2.00e+00 cbd=2.00e-14
+               cbs=2.00e-14 is=1.00e-14 pb=0.700 cgso=2.44e-10
+               cgdo=2.44e-10 cgbo=2.00e-12 rsh=75.000 cj=1.54e-04
+               mj=0.500 cjsw=4.37e-10 mjsw=0.500 js=4.19e-10 tox=8.50e-08
+               nsub=1.98e+15 tpg=1.000 xj=9.00e-07 ld=6.00e-07 uo=240.000
+               ucrit=6.44e+04 uexp=0.139 vmax=7.33e+04 xqc=0.400)
.model enn nmos(level=2 vto=0.900 kp=3.05e-05 gamma=1.592 phi=0.695
+               lambda=1.00e-02 rd=2.00e+00 rs=2.00e+00 cbd=2.00e-14
+               cbs=2.00e-14 is=1.00e-14 pb=0.700 cgso=2.84e-10
+               cgdo=2.84e-10 cgbo=2.00e-12 rsh=15.000 cj=3.44e-04
+               mj=0.500 cjsw=1.09e-09 mjsw=0.500 js=1.37e-05 tox=8.50e-08
+               nsub=9.92e+15 tpg=1.000 xj=1.00e-06 ld=7.00e-07 uo=750.000
+               ucrit=1.23e+05 uexp=0.022 vmax=4.92e+05 xqc=0.400)
* invert input 2, output 3, vdd 1
.subckt inv 2 3 1
m1u 3 2 1 1 enp w=15u l=5u ad=75p as=75p pd=40u ps=40u
m1p 3 2 0 0 enn w=5u l=5u ad=25p as=25p pd=20u ps=20u
.ends inv
* input 3, output 4, vdd 1
.subckt load1 3 4 1
x1 3 4 1 inv
x2 4 5 1 inv
cload1 5 0 0.1pf
.ends load1
* input 3, outout 4, vdd 1, for all follows
.subckt load2 3 4 1
x1 3 4 1 inv
x2 3 5 1 inv
x3 4 6 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
.ends load2
.subckt load3 3 4 1
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 4 7 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
.ends load3
.subckt load4 3 4 1
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 4 8 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
```

```
cload3 7 0 0.1pf
cload4 8 0 0.1pf
.ends load4
.subckt load5 3 4 1
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 4 9 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
.ends load5
.subckt load6 3 4 1
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 3 9 1 inv
x7 4 10 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
cload6 10 0 0.1pf
.ends load6
.subckt load7 3 4 1
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 3 9 1 inv
x7 3 10 1 inv
x8 4 11 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
cload6 10 0 0.1pf
cload7 11 0 0.1pf
.ends load7
.subckt load8 3 4 1
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
```

```
x6 3 9 1 inv
x7 3 10 1 inv
x8 3 11 1 inv
x9 4 12 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
cload6 10 0 0.1pf
cload7 11 0 0.1pf
cload8 12 0 0.1pf
.ends load8
.subckt load9 3 4 1
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 3 9 1 inv
x7 3 10 1 inv
x8 3 11 1 inv
x9 3 12 1 inv
x10 4 13 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
cload6 10 0 0.1pf
cload7 11 0 0.1pf
cload8 12 0 0.1pf
cload9 13 0 0.1pf
.ends load9
.subckt load10 3 4 1
x1 3 4 1 inv
x2 3 5 1 inv
x3 3 6 1 inv
x4 3 7 1 inv
x5 3 8 1 inv
x6 3 9 1 inv
x7 3 10 1 inv
x8 3 11 1 inv
x9 3 12 1 inv
x10 3 13 1 inv
x11 4 14 1 inv
cload1 5 0 0.1pf
cload2 6 0 0.1pf
cload3 7 0 0.1pf
cload4 8 0 0.1pf
cload5 9 0 0.1pf
cload6 10 0 0.1pf
cload7 11 0 0.1pf
cload8 12 0 0.1pf
```

```
cload9 13 0 0.1pf
cload10 14 0 0.1pf
.ends load10
*  call subcircuits
x0 3 4 1 load1
x1 3 6 1 load2
x3 3 8 1 load3
x4 3 10 1 load4
x5 3 12 1 load5
x6 3 14 1 load6
x7 3 16 1 load7
x8 3 18 1 load8
x9 3 20 1 load9
x10 3 22 1 load10
vdd 1 0 dc 5v
vin 3 0 pulse (0 5 0ns 0ns 0ns 20ns 40ns)
.options abstol=1.e-12 vntol=1.e-12 limpts=3000
.tran 0.01ns 25ns
.plot tran  v(3) v(4) v(5) v(6) v(8) v(10) v(14) v(16) (0,5)
.plot tran  v(3) v(18) v(20) (0,5)
.end
```

APPENDIX D

Implementation Package

```
/* This a main program of layout generator.
zigzag, pal, star, fun, fun0, pmos, nmos, kicgen, cifgen will be called */
#include <sgtty.h>
#include <strings.h>
#include <stdio.h>
#include <math.h>
FILE *fp, *fopen();
main()
{
 FILE *ptr, *fopen();
 int height;
 int axis2=0;
 int i, n, r, s, wn;
 int in, per;

 char name[20], name1[20], name2[20], ca, ch, type, key, ret, yes;

 float beta, Y, fr, wn1, wn2, total_width, thre, tau0, delay1, delay2;
 float an=0.2, ap=0.2, a, bn=0.149, bp=0.149, b, kn, cload, cgn, cg, se;
 float dt1, dt2, dt3, dt4, dt, scale=3.37;
 float delta1, delta2, delta3, delta4, delta5;
 float gamma1, gamma2, gamma3;
 float g1, g2, g3;
 float , lambda, optimum_stage, expression, expression1, expression2;

/* spice parameters */
   float CGDOn=2.84e-10, CGSOn=2.84e-10, CJn=3.44e-4, CJSWn=1.09e-9;
   float CGDOp=2.44e-10, CGSOp=2.44e-10, CJp=1.54e-4, CJSWp=4.37e-10;
   float cox=4.06e-4, un=750, up=240, LDn=7e-7, LDp=6e-7;
   float keq=0.52; /* for PB=0.7v, MJ=MJCW=0.5 */

/* set screen */

system ("clear");

 printf("    ************************************************************\n");
 printf("    *                                                          *\n");
 printf("    *                                                          *\n");
 printf("    *                                                          *\n");
 printf("    *              CMOS-1B BUFFER GENERATOR                    *\n");
 printf("    *               IN MULTI-CONFIGURATION                     *\n");
 printf("    *                                                          *\n");
 printf("    *                  version 1.1                             *\n");
 printf("    *                                                          *\n");
 printf("    *                                                          *\n");
 printf("    *                                                          *\n");
 printf("    *                                                          *\n");
```

```
printf("    *                Written by Y. Zhu                    *\n");
printf("    *          Electrical and Computer Department          *\n");
printf("    *                Concordia University                 *\n");
printf("    *                  May, 1987                          *\n");
printf("    *                                                     *\n");
printf("    *                                                     *\n");
printf("    *                                                     *\n");
printf("    *********************************************************************\n");
printf("\n");
printf("                hit return key to continue ..\n");
scanf("%c", &key);

system ("clear");
/* refresh(); */

   printf("\nDo you want to generate a driver as an output PAD ? (y/n)\n");
   scanf("%c", &yes);

   printf("\nLoad capacitance (default unit pf) =?");
   scanf("%f", &cload);

   printf("\nMicrons per lambda ?");
   scanf("%f", &lambda);

   printf("\nChoose the ratio of channel width of p- and n-transistor \n");
   printf("and the same length for both transistors are assumed. \n?");
   scanf("%f", &beta);
{
   l=2e-6*lambda; /* m */
   wn1=2e-6*lambda; /* m */
   kn=cox*750e-4*(wn1/l);
   cgn=wn1*l*cox; /* F */
   delta1=1+2*LDn/l;
   delta2=1+2*LDp/l;
   delta3=CJp/CJn;
   delta4=CJSWp/CJSWn;
   delta5=delta4;
   gamma1=(keq*CJn)/cox;
   gamma2=2*(keq*CJSWn)/(cox*l);
   gamma3=gamma2;
   g1=(1+delta3*beta)/(delta1+delta2*beta);
   g2=(1+delta4*beta)/(delta1+delta2*beta);
   g3=(1+delta5)/(delta1+delta2*beta);

   a=(an+ap*un/(beta*up))/2;
   b=(bp+bn*un/(beta*up))/2;

   cg=cgn*(delta1+delta2*beta); /* F */

   tau0=(cg/kn)*(a+b);

   Y=1e-12*cload/cg;
   if (Y<5.0)
     {
```

```
    printf("\nThe fan-out of buffer is less than 5 !\n");
    exit(0);
   }

if (yes=='y')
  {
  n=2;
  printf("\nThis CMOS PAD has 2 stages.\n");
  }

else
  {
   printf("\nSelect a letter from following to optimize your buffer");
   printf("in speed, power or area\n");
   printf("\ns       - speed\n");
   printf("p       - power\n");
   printf("a       - area\n");

letter_detective1:
   scanf("%c", &ch);
   ch=getchar();
   if ( ch=='s' )
      goto speed_optimum;
   else if ( ch=='p' || ch=='a' )
      goto power_area_optimum;
  else
     {
      printf("\nThe input letter is invalid ! try again.\n");
      goto letter_detective1;
     }

speed_optimum:
   optimum_stage = log(Y)/log(scale);
   r = (int) optimum_stage;
   fr = ( float ) r;
   expression1=r*(pow(Y, 1/fr)+g1*gamma1+g2*gamma2+g3*gamma3);
   expression2=(r+1)*(pow(Y, 1/(fr+1))+g1*gamma1+g2*gamma2+g3*gamma3);
   if ( expression1 < expression2 )
     n=r;
   else
     n=r+1;
   goto type_select;

power_area_optimum:
   dt2=(scale+g1*gamma1+g2*gamma2+g3*gamma3)/(scale*log(scale)*log(scale)) ;
   dt2 = (1e-12*cload*log(Y)*(a+b)*(1/log(scale)-dt2))/(Y*kn) ;
   dt1 = dt2 ;
   dt4 = dt2/dt1 ;
loop_1:
   scale = scale + 0.01 ;
   dt1 = dt2 ;
   dt3 = dt4 ;
   dt2=(scale+g1*gamma1+g2*gamma2+g3*gamma3)/(scale*log(scale)*log(scale)) ;
   dt2 = (1e-12*cload*log(Y)*(a+b)*(1/log(scale)-dt2))/(Y*kn) ;
```

```
    if (dt2 <= 0 )
      goto loop_1;
    else
      {
      dt4 = dt2/dt1 ;
      dt = dt3 - dt4 ;
      if ( dt < 0 )
        dt = (-1) * dt ;
      }
    if ( dt > 1e-5)
      goto loop_1;

    optimum_stage = log(Y)/log(scale) ;
    r = (int) optimum_stage ;
    n = r ;
    }

  type_select:

    printf("\nSelect the layout configuration of poly routing from following\n");
    printf("\nz      - zigzag\n");
    printf("p      - parallel\n");
    printf("s      - star cross\n");

letter_detective3:
    scanf("%c", &type);
    type=getchar();
    if (type!='z' && type!='p' && type!='s')
      {
      printf("The input letter is invalid ! try again.\n");
      goto letter_detective3;
      }

    printf("\nSelect your output file from following\n");
    printf("\nc      - cif file\n");
    printf("k      - kic file\n");
    printf("b      - both of cif and kic file\n");

label1:
    scanf("%c", &ch);
    ch=getchar();
    if (ch!='c' && ch!='k' && ch!='b')
      {
      printf("The input letter is invalid ! try again.\n");
      goto label1;
      }
    if (ch=='b')
      {
      printf("\nThe output cif file name ?\n");
      scanf("%s", name1);
      printf("\nThe output kic file name ?\n");

label2:
    scanf("%s", name2);
```

```
    if (strcmp(name1, name2) == 0)
     {
      printf("\nKic file name is same as cif file ! rename it please.\n");
      goto label2;
     }
   }

 else
  {
   printf("Output file name ?");
   scanf("%s", name);
  }

 printf("\nRuning ...\n");
 printf("\nThis CMOS driver has %d stages. \n", n );
 se=log(Y)/n;
 scale=exp(se); /* readjust scaling factor */
 s=(int) 100*lambda;

 ptr=fopen("data", "a");
 for (i=1; i<n; i++)
  {
   wn2=1e-6*cload/(pow(scale, ((floaf) n-i+1))*cox*l*(delta1+delta2*beta)); /* um */
   wn=(int) 100*wn2;
   total_width=total_width + wn2;
   fprintf(ptr, "%d ", wn);
  }
 fclose(ptr);

 height=(int) (100*total_width)/n;

 if (type=='z')
   {
    fp=fopen("data", "r");
    for (i=0; i<n; ++i) {
      axis2=zigzag(height, beta, axis2, s);
    }
    fclose (fp);
    fun0(s, axis2);
   }
 else if (type=='p')
   {
    fp=fopen("data","r");
    for (i=0; i<n; ++i) {
      axis2=pal(height, beta, axis2, s);
    }
    fclose (fp);
    fun(s);
   }
 else if (type=='s')
    star(beta, wn, s);
}

 if (ch=='c')
```

```
        cifgen(name);
      else if (ch=='k')
        kicgen(name);
       else if (ch=='b')
          {
           cifgen(name1);
           kicgen(name2);
          }

          unlink("data");
          unlink("Poly_box");
          unlink("LCF_box");
          unlink("LCPP_box");
          unlink("LCNP_box");
          unlink("LCC_box");
          unlink("LCM_box");
          unlink("LCPW_box");
          unlink("LCPG_box");
          unlink("LCNG_box");
printf("Okay, the job is done !\n");
}

/* A module to auto generate zigzag structure inverter,
Poly routing in vertical direction */
#include <stdio.h>
#include <math.h>
int wycf, nxcf, nwxcf, nwycf, nycf;
extern FILE *fp;
zigzag(height, beta, axis2, s)
float beta;
int height, axis2, s;
{
 int x=0, y=0;
 int wn, wp;
 int threshold, thres;

 int c, i, j;
 FILE *ptr, *fopen();
 char b='B';
 char q=';';
 int wxbp, wxsp, wxcf, wxpp, wxnp, wxcc, wxcm;
 int wybp, wysp, wypp, wynp, wycc, wycm;
 int xbp, xsp, xcf, xcc, xcm, xpg, xng;
 int ybp, ysp, ycf, ycc, ycm, ypg, yng;
 int wxccu, wyccu,xccu, yccu, wxcm1, wycm1, xcm1, ycm1;
 int wxcfi, wycfi, xcfi, ycfi, wxcfu, wycfu, xcfu, ycfu;
 int wxpbi, wypbi, xpbi, ypbi;
 int n1, n2, m1, m2, ms1, ms2;

 int nwxbp, nwxsp, nwxpp, nwxnp, nwxcc, nwxcm, nwxpg, nwxng;
 int nwybp, nwysp, nwypp, nwynp, nwycc, nwycm, nwypg, nwyng;
 int nxbp, nxsp, nxcc, nxcm, nxpp, nxnp;
 int nybp, nysp, nycc, nycm, nypp, nynp;
```

```
int nwxcc1, nwycc1, nxcc1, nycc1, nwxcm1, nwycm1, nxcm1, nycm1;
int wxcml, wycml, xcml, ycml, wxcmu, wycmu, xcmu, ycmu;
int nwxcfl, nwycfl, nxcfl, nycfl, nwxcfu, nwycfu, nxcfu, nycfu;
int nwxpbl, nwypbl, nxpbl, nypbl;
int p, offset, n;

int wxpl, wypl, xpl, ypl, nwxpl, nwypl, nxpl, nypl;
int wxcmg, wycmg, xcmg, ycmg, wxcmo, wycmo, xcmo, ycmo;
int wxcmv, wycmv, xcmv, ycmv;
int wxcc1, wycc1,xcc1, ycc1;
float a=0.2, yspf, nyspf;

int wxpe, wype, xpe, ype, nwxpe, nwype, nxpe, nype;
int wxbpl, wybpl, xbpl, ybpl, wxbpe, wybpe, xbpe, ybpe;
int wxcco, wycco, xcco, ycco, wxbpo, wybpo, xbpo, ybpo;

 offset=2*s;
 x=s+abs(axis2);

fscanf(fp, "%d", &c);
wn=c;

wp=beta*wn;

if (wp%2==1) /* w=odd */
  {
   wp=wp+1;
  }
if (wn%2==1)
  {
   wn=wn+1;
  }


threshold = a*beta*height;

thres=(int) beta*4*s+2*s; /* the minimun height of vertical poly */

if (threshold <= thres)
  {
   threshold=thres-2*s;
  }

n1=wp/(threshold+2*s);

if ( wp <= threshold+5*s)
  j=1;
else
  j=2;
switch(j)
{
case 1:
  wxcf=pmos(wp, threshold, axis2, beta, s);
  break;
```

```
case 2:
  { /* ptransistor */


    ptr = fopen("Poly_box", "a");

    for(i=0; i<n1+1; ++i)
     {
      wxbp=2*s;
      wybp=threshold;
      xbp=i*4*s+x;
      ybp=8*s+wybp/2+y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxbp, wybp, xbp, ybp, q);
     }
/* small poly box */

    for(i=0; i<n1; ++i)
     {
      wxsp=2*s;
      wysp=2*s;
      xsp=offset+i*4*s+x;
      yspf=pow(-1.0,(float) i);
      ysp=(((int) yspf)+1)*(wybp-2*s)/2+9*s+y;
      /* ysp=(pow(-1, i)+1)*(wybp-2*s)/2+9*s+y */
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxsp, wysp, xsp, ysp, q);
     }
      { /* last big poly box to extension */
      wxpe=4*s;
      wype=2*s;
      xpe=xsp+wxpe/2+3*s;
      if (n1%2===1)
        ype=ybp-wybp/2+wype/2;
      else
      ype=ybp+wybp/2-wype/2;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxpe, wype, xpe, ype, q);
      }
     fclose(ptr);

/* device well box */
    ptr = fopen("LCF_box", "a");
     {
     wxcf=4*s*(n1+1)+2*s;
     wycf=wybp+14*s;
     xcf=wxcf/2-3*s+x;
     ycf=3*s+wycf/2+y;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxcf, wycf, xcf, ycf, q);
     }
    fclose(ptr);

/* p-pulse box */
/*    ptr = fopen("LCPP_box", "a");
     {
     wxpp=wxcf+6*s;
     wypp=wycf-2*s;
```

```
        xpp=xcf+s;
        ypp=s+wypp/2+y;
       fprintf(ptr, "%c %d %d %d %d%c0, b, wxpp, wypp, xpp, ypp, q);
        }
       fclose(ptr);
 */
 /* n-pulse box */
 /*     ptr = fopen("LCNP_box", "a");
        {
        wxnp=wxpp+4*s;
        wynp=wypp+2*s;
        xnp=xpp;
        ynp=ypp-s;
       fprintf(ptr, "%c %d %d %d %d%c0, b, wxnp, wynp, xnp, ynp, q);
        }
       fclose(ptr);
 */
       m1=wxcf/(7*s);
       ms1=(wxcf-4*s)/(7*s);

 /* contact cut box */
       ptr = fopen("LCC_box", "a");
       /* upper cut box */
       for(i=0; i<m1; ++i)
        {
         wxcc1=2*s;
         wycc1=6*s;
         xcc1=s+i*(7*s)+x;
         ycc1=wycf-s+y;
         fprintf(ptr, "%c %d %d %d %d%c0, b, wxcc1, wycc1, xcc1, ycc1, q);
        }
       /* below cut box */
       for(i=0; i<ms1; ++i)
        {
         wxcc=2*s;
         wycc=2*s;
         xcc=4*s+i*7*s+x;
         ycc=5*s+y;
         fprintf(ptr, "%c %d %d %d %d%c0, b, wxcc, wycc, xcc, ycc, q);
        }
       fclose(ptr);

 /* metal box */
       ptr = fopen("LCM_box", "a");
       /* upper metal */
       for(i=0; i<m1; ++i)
        {
         wxcm1=2*s+wxcc1;
         wycm1=2*s+wycc1;
         xcm1=s+i*7*s+x;
         ycm1=ycc1;
         fprintf(ptr, "%c %d %d %d %d%c0, b, wxcm1, wycm1, xcm1, ycm1, q);
        }
       /* below metal */
```

```
    for(i=0; i<ms1; ++i)
     {
      wxcm=wxcc+2*s;
      wycm=wycc+2*s;
      xcm=4*s+i*7*s+x;
      ycm=ycc;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcm, wycm, xcm, ycm, q);
     }
    fclose(ptr);
    }
    break;
  } /* switch loop */

threshold = a*height;

if (threshold <= 6*s)
  {
   threshold = 4*s;
  }

n2=wn/(threshold+2*s);

if ( wn <= threshold+5*s)
  j=3;
else
  j=4;

switch(j)
  {
  case 3:
    nmos(wn, threshold, axis2, k, s);
    break;

  case 4:
    {/* ntrsistor */


      ptr = fopen("Poly_box", "a");
     /* big poly box */
     for(i=0; i<n2+1; ++i)
      {
       nwxbp=2*s;
       nwybp=threshold;
       nxbp=i*4*s+x;
       nybp=(-1)*(nwybp/2+12*s)+y;
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxbp, nwybp, nxbp, nybp, q);
      }
     /* small poly box */
     for(i=0; i<n2; ++i)
      {
       nwxsp=2*s;
       nwysp=2*s;
       nxsp=2*s+i*4*s+x;
       nyspf=pow(-1.0, ((float) i));
```

```
        nysp=(((int) nyspf)+1)*(2*s-nwybp)/2-13*s+y;
        /* nysp=(pow*(-1, 1)+1)*(2*s-nwybp)/2-13*s+y */
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxsp, nwysp, nxsp, nysp, q);
       }
        { /* ntrsistor big poly extension */
        nwxpe=4*s;
        nwype=2*s;
        nxpe=nxsp+nwxpe/2+3*s;
        if (n2%2==0)
           nype=nybp-nwybp/2+nwype/2;
        else
        nype=nybp+nwybp/2-nwype/2;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxpe, nwype, nxpe, nype, q);
        }
      fclose(ptr);

/* device-well box */
     ptr = fopen("LCF_box", "a");
      {
      nwxcf=4*s*(n2+1)+2*s;
      nwycf=nwybp+14*s;
      nxcf=nwxcf/2-3*s+x;
      nycf=(-1)*(nwycf/2+7*s)+y;
     fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcf, nwycf, nxcf, nycf, q);
      }
     fclose(ptr);

/* p-well box */
/*     ptr = fopen("LCPW_box", "a");
      {
      nwxpw=nwxcf+4*s;
      nwypw=nwycf+4*s;
      nxpw=nxcf;
      nypw=nycf;
     fprintf(ptr, "%c %d %d %d %d%c0, b, nwxpw, nwypw, nxpw, nypw, q);
      }
     fclose(ptr);
*/
/* p-gard box */
/*     ptr = fopen("LCPG_box", "a");
      {
      nwxpg=nwxpw+4*s;
      nwypg=nwypw+4*s;
      nxpg=nxcf;
      nypg=nycf;
     fprintf(ptr, "%c %d %d %d %d%c0, b, nwxpg, nwypg, nxpg, nypg, q);
      }
     fclose(ptr);
*/
/* n-gard box */
/*     ptr = fopen("LCNG_box", "a");
      {
      nwxng=nwxpg+4*s;
      nwyng=nwypg+4*s;
```

```
        nxng=nxcf;
        nyng=nycf;
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxng, nwyng, nxng, nyng, q);
        }
      fclose(ptr);
*/
      m2=nwxcf/(7*s);
      ms2=(nwxcf-4*s)/(7*s);

/* contact cut box */
      ptr = fopen("LCC_box", "a");
      /* upper contact */
      for(i=0; i<ms2; ++i)
       {
        nwxcc1=2*s;
        nwycc1=2*s;
        nxcc1=4*s+i*7*s+x;
        nycc1=(-1)*9*s+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcc1, nwycc1, nxcc1,
              nycc1, q);
        }
       /* below contact */
      for(i=0; i<m2; ++i)
       {
        nwxcc=2*s;
        nwycc=6*s;
        nxcc=s+i*7*s+x;
        nycc=(-1)*(nwycf+3*s)+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcc, nwycc, nxcc, nycc, q);
       }
      fclose(ptr);

/* metal box */
      ptr = fopen("LCM_box", "a");
      /* upper contact */
      for(i=0; i<ms2; ++i)
       {
        nwxcm1=nwxcc1+2*s;
        nwycm1=nwycc1+2*s;
        nxcm1=4*s+i*7*s+x;
        nycm1=nycc1;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcm1, nwycm1, nxcm1,
              nycm1, q);
        }
      /* below metal box */
      for(i=0; i<m2; ++i)
       {
        nwxcm=2*s+nwxcc;
        nwycm=nwycc+2*s;
        nxcm=s+i*7*s+x;
        nycm=nycc;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcm, nwycm, nxcm, nycm, q);
       }
      fclose(ptr);
```

```
/* p-pulse box */
      ptr = fopen("LCPP_box", "a");
       {
       nwxpp=nwxcf+4*s;
       nwypp=6*s;
       nxpp=nxcf;
       nypp=nycc-nwypp/2;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxpp, nwypp, nxpp, nypp, q);
       }
      fclose(ptr);

/* n-pulse box */
      ptr = fopen("LCNP_box", "a");
       {
       nwxnp=nwxpp+4*s;
       nwynp=nwypp+2*s;
       nxnp=nxpp;
       nynp=nycc-nwynp/2;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxnp, nwynp, nxnp, nynp, q);
       }
      fclose(ptr);
     }
     break;
  } /* switch loop */
/* output metal box */
   ptr = fopen("LCM_box", "a");
   {
    wxcmo=wxcf+2*s;
    wycmo=10*s;
    xcmo=wxcmo/2+2*s+x;
    ycmo=(-1)*(wycmo/2-3*s)+y;
   fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmo, wycmo, xcmo, ycmo, q);
   }
   fclose(ptr);

/* output cut box */
   ptr=fopen("LCC_box", "a");
   {
    wxcco=2*s;
    wycco=2*s;
    xcco=wxcmo+x;
    ycco=y;
   fprintf(ptr, "%c %d %d %d %d%c0, b, wxcco, wycco, xcco, ycco, q);
   }
   fclose(ptr);

/* p-n connection poly */
   ptr = fopen("Poly_box", "a");
    {
    wxbpl=2*s;
    wybpl=22*s;
    xbpl=x;
    ybpl=(-1)*(wybpl/2-8*s)+y;
   fprintf(ptr, "%c %d %d %d %d%c0, b, wxbpl, wybpl, xbpl, ybpl, q);
```

```
    }
/* input poly */
    {
    wxbpe=4*s;
    wybpe=2*s;
    xbpe=x-wxbpe/2-s;
    ybpe=y;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxbpe, wybpe, xbpe, ybpe, q);
    }
/* output contact poly box */
    {
    wxbpo=6*s;
    wybpo=6*s;
    xbpo=xcco;
    ybpo=ycco;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxbpo, wybpo, xbpo, ybpo, q);
    }
    fclose(ptr);

   x=x+wxcf+9*s;
   return(x);
}


/* A module to auto generate parallel structure inverter */
#include <stdio.h>
#include <math.h>
int n1, m1, ycc1, nwycf, xcmo, nxcc, nycc;
extern FILE *fp;
pal(height, beta, axis2, s)
float beta;
int height, axis2, s;
{
 int wycf, nxcf, nwxcf,  nycf;
 int x=0, y=0;
 int wn, wp;
 int threshold, thres;

  int c, i, j;
  FILE *ptr, *fopen();
  char b='B';
  char q=';';
  int wxbp, wxsp, wxcf, wxpp, wxnp, wxcc, wxcm;
  int wybp, wysp, wypp, wynp, wycc, wycm;
  int xbp, xsp, xcf, xcc, xcm, xpg, xng;
  int ybp, ysp, ycf, ycc, ycm, ypg, yng;
  int wxccu, wyccu,xccu, yccu, wxcm1, wycm1, xcm1, ycm1;
  int wxcfi, wycfi, xcfi, ycfi, wxcfu, wycfu, xcfu, ycfu;
  int wxpbi, wypbi, xpbi, ypbi;
  int n2, m2, ms1, ms2;

  int nwxbp, nwxsp, nwxpp, nwxnp, nwxcc, nwxcm, nwxpg, nwxng;
  int nwybp, nwysp, nwypp, nwynp, nwycc, nwycm, nwypg, nwyng;
  int nxbp, nxsp, nxcm, nxpp, nxnp;
```

```
int nybp, nysp, nycm, nypp, nynp;
int nwxcc1, nwycc1, nxcc1, nycc1, nwxcm1, nwycm1, nxcm1, nycm1;
int wxcml, wycml, xcml, ycml, wxcmu, wycmu, xcmu, ycmu;
int nwxcfl, nwycfl, nxcfl, nycfl, nwxcfu, nwycfu, nxcfu, nycfu;
int nwxpbl, nwypbl, nxpbl, nypbl;
int p, offset1, offset2, n;

int wxpl, wypl, xpl, ypl, nwxpl, nwypl, nxpl, nypl;
int wxcmg, wycmg, xcmg, ycmg, wxcmo, wycmo, ycmo;
int wxcmv, wycmv, xcmv, ycmv;
int wxcc1, wycc1,xcc1;
float a=0.2, yspf, nyspf;

int wxpe, wype, xpe, ype, nwxpe, nwype, nxpe, nype;
int wxbpl, wybpl, xbpl, ybpl, wxbpe, wybpe, xbpe, ybpe;
int wxcco, wycco, xcco, ycco, wxbpo, wybpo, xbpo, ybpo;
int wxccm, wyccm, xccm, yccm, wxcml, wycml, xcml, ycml;
int nwxcml, nwycml, nxcml, nycml;

 offset1=10*s;
 offset2=5*s;
 x=abs(axis2);
/* printf("x=%d0, x);
*/
 fscanf(fp, "%d", &c);
 wn=c;

 wp=beta*wn;

 if (wp%2==1) /* w=odd */
  {
   wp=wp+1;
  }
 if (wn%2==1)
  {
   wn=wn+1;
  }


 threshold = a*beta*height;

 thres=(int) beta*7*s; /* the minimun height of vertical poly */

 if (threshold <= thres)
   {
    threshold=thres;
   }

  n1=wp/(threshold);
  if (n1<1)
   n1=1;

  { /* ptransistor */
```

```
    ptr = fopen("Poly_box", "a");

    for(i=0; i<n1; ++i)
     {
      wxbp=2*s;
      wybp=threshold+4*s;
      xbp=offset2+i*10*s+x;
      ybp=3*s+wybp/2+y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxbp, wybp, xbp, ybp, q);
     }
/* middle poly box */

    for(i=0; i<n1; ++i)
     {
      wxsp=6*s;
      wysp=6*s;
      xsp=offset2+i*10*s+x;
      ysp=y;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxsp, wysp, xsp, ysp, q);
    }
     fclose(ptr);

/* device well box */
     ptr = fopen("LCF_box", "a");
     for (i=0; i<n1; ++i)
      {
      wxcf=wxbp+4*s;
      wycf=threshold;
      xcf=offset2+i*10*s+x;
      ycf=ybp;
     fprintf(ptr, "%c %d %d %d %d%c0, b, wxcf, wycf, xcf, ycf, q);
      }
     fclose(ptr);

     if (n1%2==0)
      {
       m1=(n1+2)/2;
       ms1=n1/2;
      }
     else
      {
       m1=(n1+1)/2;
       ms1=m1;
      }

/* contact cut box */
     ptr = fopen("LCC_box", "a");
     /* upper cut box */
     for(i=0; i<m1; ++i)
      {
       wxcc1=2*s;
       wycc1=6*s;
       xcc1=i*(20*s)+x;
       ycc1=wybp+6*s+y;
```

```
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxcc1, wycc1, xcc1, ycc1, q);
       }
     /* below cut box */
     for(i==0; i<ms1; ++i)
      {
       wxcc==2*s;
       wycc==2*s;
       xcc==offset1+i*20*s+x;
       ycc==7*s+y;
       fprintf(ptr, "%c %d %d %d %d%c0, b, wxcc, wycc, xcc, ycc, q);
      }
     /* middle cut box */
     for (i==0; i<n1; ++i)
      {
       wxccm==2*s;
       wyccm==2*s;
       xccm==offset2+i*10*s+x;
       yccm==y;
       fprintf(ptr, "%c %d %d %d %d%c0, b, wxccm, wyccm, xccm,
            yccm, q);
      }
     fclose(ptr);

/* metal box */
     ptr == fopen("LCM_box", "a");
     /* upper metal */
     for(i==0; i<m1; ++i)
      {
       wxcm1==2*s+wxcc1;
       wycm1==2*s+wycc1;
       xcm1==i*20*s+x;
       ycm1==ycc1;
       fprintf(ptr, "%c %d %d %d %d%c0, b, wxcm1, wycm1, xcm1, ycm1, q);
      }
     /* below metal */
/*     for(i==0; i<ms1; ++i)
      {
       wxcm==wxcc+2*s;
       wycm==wycc+2*s;
       xcm==offset1+i*20*s+x;
       ycm==ycc;
       fprintf(ptr, "%c %d %d %d %d%c0, b, wxcm, wycm, xcm, ycm, q);
      }
*/
/* lower output metal */
     {
     if (n1%2===1)
     wxcml==(ms1-1)*20*s+6*s;
     else
     wxcml==(ms1-1)*20*s+16*s;
     wycml==wycc+2*s;
     xcml==wxcml/2+8*s+x;
     ycml==ycc;
     fprintf(ptr, "%c %d %d %d %d%c0, b, wxcml, wycml, xcml, ycml, q);
```

```
    }
    fclose(ptr);

/* upper contact device well */
    ptr=fopen("LCF_box", "a");
    for(i=0; i<m1; ++i)
     {
      wxcfu=wxcm1;
      if (threshold > thres)
        wycfu=wycm1+2*s;
      else
        wycfu=wycm1+3*s;
      xcfu=i*20*s+x;
      ycfu=ycm1-(wycfu-wycm1)/2;
     fprintf(ptr, "%c %d %d %d %d%c0, b, wxcfu, wycfu, xcfu, ycfu, q);
      }

/* lower contact device well */
     for (i=0; i<ms1; ++i)
       {
       wxcfl=wxcc+2*s;
       wycfl=wycc+2*s;
       xcfl=offset1+i*20*s+x;
       ycfl=ycc;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcfl, wycfl, xcfl, ycfl, q);
       }
      fclose(ptr);
  }

 threshold = a*height;

 if (threshold < 7*s)
   {
   threshold = 7*s;
   }

    {/* ntrsistor */

      ptr = fopen("Poly_box", "a");
     /* big poly box */
     for(i=0; i<n1; ++i)
       {
        nwxbp=2*s;
        nwybp=threshold+4*s;
        nxbp=offset2+i*10*s+x;
        nybp=(-1)*(nwybp/2+3*s)+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxbp, nwybp, nxbp, nybp, q);
       }
      fclose(ptr);

/* device-well box */
     ptr = fopen("LCF_box", "a");
      for (i=0; i<n1; ++i)
       {
```

```
        nwxcf=4*s+nwxbp;
        nwycf=threshold;
        nxcf=offset2+i*10*s+x;
        nycf=nybp;
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcf, nwycf, nxcf, nycf, q);
        }
     fclose(ptr);

/* contact cut box */
     ptr = fopen("LCC_box", "a");
     /* upper cut */
     for(i=0; i<ms1; ++i)
      {
       nwxcc1=2*s;
       nwycc1=2*s;
       nxcc1=offset1+i*20*s+x;
       nycc1=(-1)*ycc;
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcc1, nwycc1, nxcc1,
            nycc1, q);
       }

     /* below cut */
     for(i=0; i<m1; ++i)
      {
       nwxcc=2*s;
       nwycc=6*s;
       nxcc=i*20*s+x;
       nycc=(-1)*(nwybp+6*s)+y;
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcc, nwycc, nxcc, nycc, q);
      }
     fclose(ptr);

/* metal box */
     ptr = fopen("LCM_box", "a");
     /* upper output */
      {
       nwxcm=wxcml;
       nwycm=wycml;
       nxcm=xcml;
       nycm=(-1)*(ycml);
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcm, nwycm, nxcm,
            nycm, q);
      }

     /* below metal box */
     for (i=0; i<m1; ++i)
      {
       nwxcml=2*s+nwxcc;
       nwycml=nwycc+2*s;
       nxcml=i*20*s+x;
       nycml=nycc;
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcml, nwycml, nxcml,
            nycml, q);
      }
```

```
    fclose(ptr);

/* below CF box */
    ptr=fopen("LCF_box", "a");
    for (i=0; i<m1; ++i)
      {
       nwxcfl=nwxcml;
       if (threshold > 7*s)
         nwycfl=nwycml+2*s;
       else
         nwycfl=nwycml+3*s;
       nxcfl=i*20*s+x;
       nycfl=nycml+(nwycfl-nwycml)/2;
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcfl, nwycfl, nxcfl,
            nycfl, q);
      }

/* upper CF box */
    for (i=0; i<ms1; ++i)
      {
       nwxcfu=nwxcc1+2*s;
       nwycfu=nwycc1+2*s;
       nxcfu=offset1+i*20*s+x;
       nycfu=nycc1;
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcfu, nwycfu, nxcfu,
            nycfu, q);
      }
    fclose(ptr);

/* input metal box */
  ptr=fopen("LCM_box", "a");
  {
   wxcmi=n1*(wxsp+4*s)+s;
   wycmi=4*s;
   xcmi=(wxcmi)/2-4*s+x;
   ycmi=y;
   fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmi, wycmi, xcmi, ycmi, q);
  }

/* output metal box */
  {
   wxcmo=4*s;
   wycmo=10*s;
   xcmo=xccm+5*s+wxcmo/2;
   ycmo=y;
  }
  fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmo, wycmo, xcmo, ycmo, q);
  fclose(ptr);
 }

  if (n1%2==0)
    x=xcmo+6*s;
  else
    x=xcmo+5*s;
```

```
  return(x);
}

/* A module to auto generate star structure inverter */
#include <stdio.h>
#include <math.h>
star(beta, wn, s)
float beta;
int wn, s;
{
 int x=0, y=0;
 int wp;
 int threshold, threshold_n;

 int c, i, j;
 FILE *ptr, *fopen();
 char b='B';
 char q=';';

 int wxbpe, wxbps, wxbpn, wxbpw, wxcfe, wxcfs, wxcfw, wxcfn, wxpp;
 int wxnp, wxcce, wxccs, wxccw, wxccn, wxcme, wxcms, wxcmw, wxcmn;
 int wybpe, wybps, wybpn, wybpw, wycfe, wycfs, wycfw, wycfn, wypp;
 int wynp, wycce, wyccs, wyccw, wyccn, wycme, wycms, wycmw, wycmn;
 int xbpe, xbps, xbpn, xbpw, xcfe, xcfs, xcfw, xcfn, xpp, xnp;
 int xcce, xccs, xccw, xccn, xcme, xcms, xcmw, xcmn;
 int ybpe, ybps, ybpn, ybpw, ycfe, ycfs, ycfw, ycfn, ypp, ynp;
 int ycce, yccs, yccw, yccn, ycme, ycms, ycmw, ycmn;

 int wxcm, wycm, xcm, ycm, wxcmcm, wycmcm, xcmcm, ycmcm;
 int wxccm, wyccm,xccm, yccm, wxcmm, wycmm, xcmm, ycmm;
 int wxcfm, wycfm, xcfm, ycfm, wxcfh, wycfh, xcfh, ycfh;
 int wxcfv, wycfv, xcfv, ycfv;
 int wxpbi, wypbi, xpbi, ypbi;
 int wxvdd, wyvdd, xvdd, yvdd, wxgnd, wygnd, xgnd, ygnd;

 int nwxbpe, nwxbps, nwxbpn, nwxbpw, nwxcfe, nwxcfs, nwxcfw, nwxcfn, nwxpp;
 int nwxnp, nwxcce, nwxccs, nwxccw, nwxccn, nwxcme, nwxcms, nwxcmw, nwxcmn;
 int nwybpe, nwybps, nwybpn, nwybpw, nwycfe, nwycfs, nwycfw, nwycfn, nwypp;
 int nwynp, nwycce, nwyccs, nwyccw, nwyccn, nwycme, nwycms, nwycmw, nwycmn;
 int nxbpe, nxbps, nxbpn, nxbpw, nxcfe, nxcfs, nxcfw, nxcfn;
 int nxcce, nxccs, nxccw, nxccn, nxcme, nxcms, nxcmw, nxcmn;
 int nybpe, nybps, nybpn, nybpw, nycfe, nycfs, nycfw, nycfn;
 int nycce, nyccs, nyccw, nyccn, nycme, nycms, nycmw, nycmn;

 int nwxppe, nwxpps, nwxppw, nwxppn, nwxnpe, nwxnps, nwxnpw, nwxnpn;
 int nwxpw, nwxpg, nwxng;
 int nwyppe, nwypps, nwyppw, nwyppn, nwynpe, nwynps, nwynpw, nwynpn;
 int nwypw, nwypg, nwyng;
 int nxppe, nxpps, nxppw, nxppn, nxnpe, nxnps, nxnpw, nxnpn;
 int nxpw, nxpg, nxng;
 int nyppe, nypps, nyppw, nyppn, nynpe, nynps, nynpw, nynpn;
 int nypw, nypg, nyng;
 int wxbpc, wybpc, xbpc, ybpc, nwxpe, nwype, nxpe, nype;
```

```
int nwxccm, nwyccm, nxccm, nyccm, nwxcmm, nwycmm, nxcmm, nycmm;
int nwxcfm, nwycfm, nxcfm, nycfm, nwxcfv, nwycfv, nxcfv, nycfv;
int nwxcfh, nwycfh, nxcfh, nycfh;

int wxcmcs, wycmcs, xcmcs, ycmcs, wxcmcw, wycmcw, xcmcw, ycmcw;
int wxpi, wypi, xpi, ypi, nwxpi, nwypi, nxpi, nypi;
int wxcmcv, wycmcv, xcmcv, ycmcv, wxcmo, wycmo, xcmo, ycmo;
int wxcmce, wycmce, xcmce, ycmce;

int nwxcmcn, nwycmcn, nxcmcn, nycmcn, nwxcmcw, nwycmcw, nxcmcw, nycmcw;
int nwxcm, nwycm, nxcm, nycm, nwxcmcm, nwycmcm, nxcmcm, nycmcm;
int nwxcmcg, nwycmcg, nxcmcg, nycmcg;
int nwxcmce, nwycmce, nxcmce, nycmce;


if (wn<16*s)
  wn=16*s;

wp= (int) beta*wn;

if (wp%2===1) /* wp=odd */
  {
   wp=wp+1;
  }
if (wn%2===1)
  {
   wn=wn+1;
  }

threshold = wp/4;


  { /* ptransistor */

    ptr = fopen("Poly_box", "a");
      {/* west */
       wxbpw=2*s;
       wybpw=8*s+threshold;
       xbpw=6*s+x;
       ybpw=13*s+wybpw/2+y;
       fprintf(ptr, "%c %d %d %d %d%c0, b, wxbpw, wybpw, xbpw,
             ybpw, q);
      }
      {/* south */
       wxbps=4*s+threshold;
       wybps=2*s;
       xbps=wxbps/2+7*s+x;
       ybps=14*s+y;
       fprintf(ptr, "%c %d %d %d %d%c0, b, wxbps, wybps, xbps,
       ybps, q);
      }
      {/* north */
       wxbpn=4*s+threshold;
```

```
        wybpn=2*s;
        xbpn=xbps;
        ybpn=12*s+wybpw+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxbpn, wybpn, xbpn,
            ybpn, q);
        }
        {/* east */
        wxbpe=2*s;
        wybpe=wybpw;
        xbpe=8*s+wxbps+x;
        ybpe=ybpw;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxbpe, wybpe, xbpe,
            ybpe, q);
        }
        {/* p-n connection */
        wxbpc=2*s;
        wybpc=25*s;
        xbpc=6*s+x;
        ybpc=s/2+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxbpc, wybpc, xbpc,
            ybpc, q);
        }
    fclose(ptr);

/* device well box.*/
    ptr = fopen("LCF_box", "a");
      {/* vertical */
      wxcfv=threshold;
      wycfv=2*s+wybpw;
      xcfv=xbps;
      ycfv=ybpw;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxcfv, wycfv, xcfv,
        ycfv, q);
      }
      {/* horizetal */
      wxcfh=9*s+wxbps;
      wycfh=threshold;
      xcfh=xbps+3*s/2;
      ycfh=ybpw;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxcfh, wycfh, xcfh,
        ycfh, q);
      }
      {/* west contact device */
      wxcfw=8*s;
      wycfw=4*s;
      xcfw=x;
      ycfw=ybpw;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxcfw, wycfw, xcfw,
        ycfw, q);
      }
      {/* south contact device */
      wxcfs=4*s;
      wycfs=8*s;
      xcfs=xbps;
```

```
        ycfs=8*s;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcfs, wycfs, xcfs,
            ycfs, q);
        }
        {/* north contact device */
        wxcfn=4*s;
        wycfn=8*s;
        xcfn=xbpn;
        ycfn=wycfn/2+ybpn+2*s;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcfn, wycfn, xcfn,
            ycfn, q);
        }
        {/* east contact device */
        wxcfe=4*s;
        wycfe=4*s;
        xcfe=wxcfe/2+xbpe+5*s;
        ycfe=ybpw;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcfe, wycfe, xcfe,
            ycfe, q);
        }
        {/* middle contact device */
        wxcfm=4*s;
        wycfm=4*s;
        xcfm=xcfs;
        ycfm=ybpw;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcfm, wycfm, xcfm,
            ycfm, q);
        }

      fclose(ptr);


/* contact cut box */
      ptr = fopen("LCC_box", "a");
        {/* middle cut box */
        wxccm=2*s;
        wyccm=2*s;
        xccm=xcfm;
        yccm=ycfm;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxccm, wyccm, xccm,
              yccm, q);
        }
        {/* west cut box */
        wxccw=6*s;
        wyccw=2*s;
        xccw=xcfw;
        yccw=ycfw;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxccw, wyccw, xccw,
              yccw, q);
      }
        {/* south cut box */
        wxccs=2*s;
        wyccs=6*s;
```

```
        xccs=xcfs;
        yccs=ycfs;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxccs, wyccs, xccs,
              yccs, q);
      }
       {/* north cut box */
        wxccn=2*s;
        wyccn=6*s;
        xccn=xcfn;
        yccn=ycfn;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxccn, wyccn, xccn,
              yccn, q);
      }
       {/* east cut box */
        wxcce=2*s;
        wycce=2*s;
        xcce=xcfe;
        ycce=ycfe;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxcce, wycce, xcce,
              ycce, q);
      }
    fclose(ptr);

/* metal box */
    ptr = fopen("LCM_box", "a");
       {/* middle metal */
        wxcmm=wxcfm;
        wycmm=wycfm;
        xcmm=xcfm;
        ycmm=ycfm;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmm, wycmm, xcmm,
              ycmm, q);
      }
       {/* west metal */
        wxcmw=wxcfw;
        wycmw=wycfw;
        xcmw=xcfw;
        ycmw=ycfw;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmw, wycmw, xcmw,
              ycmw, q);
      }
       {/* south metal */
        wxcms=wxcfs;
        wycms=wycfs;
        xcms=xcfs;
        ycms=ycfs;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxcms, wycms, xcms,
              ycms, q);
      }
       {/* north metal */
        wxcmn=wxcfn;
        wycmn=wycfn;
        xcmn=xcfn;
        ycmn=ycfn;
```

```
fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmn, wycmn, xcmn,
        ycmn, q);
}
{/* east metal */
wxcme=wxcfe;
wycme=wycfe;
xcme=xcfe;
ycme=ycfe;
fprintf(ptr, "%c %d %d %d %d%c0, b, wxcme, wycme, xcme,
        ycme, q);
}
{/* connection metal to south contact */
wxcmcs=8*s+(wxcfv-wxcfs)/2;
wycmcs=2*s;
xcmcs=wxcmcs/2+s+x;
ycmcs=ycfs;
fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmcs, wycmcs, xcmcs,
        ycmcs, q);
}
{/* connection metal to west contact */
wxcmcw=2*s;
wycmcw=10*s+(wycfh-wycfw)/2;
xcmcw=x;
ycmcw=wycmcw/2+7*s+y;
fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmcw, wycmcw, xcmcw,
        ycmcw, q);
}
{/* connection metal to VDD */
wxcmcv=2*s;
wycmcv=9*s+(wycfh-wycfw)/2;
xcmcv=x;
ycmcv=wycmcv/2+ycfw+2*s;
fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmcv, wycmcv, xcmcv,
        ycmcv, q);
}
{/* connection metal to east contact */
wxcmce=2*s;
wycmce=9*s+(wycfh-wycfw)/2;
xcmce=xcfe;
ycmce=wycmce/2+ycfe+2*s;
fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmce, wycmce, xcmce,
        ycmce, q);
}
{/* connection metal to middle contact */
wxcmcm=(wxcfh-wxcfm)/2-2*s;
wycmcm=2*s;
xcmcm=xcfm+2*s+wxcmcm/2;
ycmcm=ycfe;
fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmcm, wycmcm, xcmcm,
        ycmcm, q);
}
{/* p-n connection metal */
wxcm=2*s;
wycm=20*s+(wycfh-wycfm)/2;
```

```
        xcm==xbpe+s;
        ycm==wycm/2+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxcm, wycm, xcm,
              ycm, q);
        }
      fclose(ptr);

/* p-pulse box */
      ptr == fopen("LCPP_box", "a");
        {
        wxpp==wxcfw+wxcfh+wxcfe-2*s;
        wypp==wycfn+wycfv+wycfs-8*s;
        xpp==xcfm+5*s/2;
        ypp==8*s+wypp/2+y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxpp, wypp, xpp, ypp, q);
        }
      fclose(ptr);

/* n-pulse box */
      ptr == fopen("LCNP_box", "a");
        {
        wxnp==wxpp+2*s;
        wynp==wypp;
        xnp==xpp+s;
        ynp==ypp;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxnp, wynp, xnp, ynp, q);
        }
      fclose(ptr);
      }

threshold_n == wn/4;

    {/* ntrsistor */
      ptr == fopen("Poly_box", "a");
        {/* west */
        nwxbpw==2*s;
        nwybpw==8*s+threshold_n;
        nxbpw==6*s+x;
        nybpw==(-1)*(12*s+nwybpw/2)+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxbpw, nwybpw, nxbpw,
              nybpw, q);
        }
        {/* north */
        nwxbpn==4*s+threshold_n;
        nwybpn==2*s;
        nxbpn==nwxbpn/2+7*s+x;
        nybpn==(-1)*13*s+y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxbpn, nwybpn,
            nxbpn, nybpn, q);
      }
        {/* south */
        nwxbps==4*s+threshold_n;
        nwybps==2*s;
        nxbps==nxbpn;
```

```
        nybps=(-1)*(11*s+nwybpw)+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxbps, nwybps, nxbps,
              nybps, q);
       }
       {/* east */
        nwxbpe=2*s;
        nwybpe=nwybpw;
        nxbpe=8*s+nwxbpn+x;
        nybpe=nybpw;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxbpe, nwybpe, nxbpe,
              nybpe, q);
       }
      fclose(ptr);

/* device well box */
     ptr = fopen("LCF_box", "a");
       {/* vertical */
       nwxcfv=threshold_n;
       nwycfv=2*s+nwybpw;
       nxcfv=nxbps;
       nycfv=nybpw;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcfv, nwycfv, nxcfv,
            nycfv, q);
       }
       {/* horizatal */
       nwxcfh=9*s+nwxbps;
       nwycfh=threshold_n;
       nxcfh=nxbps+3*s/2;
       nycfh=nybpw;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcfh, nwycfh, nxcfh,
            nycfh, q);
       }
       {/* west contact device */
       nwxcfw=8*s;
       nwycfw=4*s;
       nxcfw=x;
       nycfw=nybpw;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcfw, nwycfw, nxcfw,
            nycfw, q);
       }
       {/* north contact device */
       nwxcfn=4*s;
       nwycfn=8*s;
       nxcfn=nxbpn;
       nycfn=y-7*s;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcfn, nwycfn, nxcfn,
            nycfn, q);
       }
       {/* south contact device */
       nwxcfs=4*s;
       nwycfs=8*s;
       nxcfs=nxbps;
       nycfs=nybps-nwycfs/2-2*s;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcfs, nwycfs, nxcfs,
```

```
            nycfs, q);
        }
        {/* east contact device */
        nwxcfe=8*s;
        nwycfe=4*s;
        nxcfe=nwxcfw/2+nxbpe+5*s;
        nycfe=nybpw;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcfe, nwycfe, nxcfe,
            nycfe, q);
        }
        {/* middle contact device */
        nwxcfm=4*s;
        nwycfm=4*s;
        nxcfm=nxcfs;
        nycfm=nybpw;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcfm, nwycfm, nxcfm,
            nycfm, q);
        }

      fclose(ptr);



/* contact cut box */
      ptr = fopen("LCC_box", "a");
        {/* middle cut box */
        nwxccm=2*s;
        nwyccm=2*s;
        nxccm=nxcfm;
        nyccm=nycfm;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxccm, nwyccm, nxccm,
              nyccm, q);
        }
        {/* west cut box */
        nwxccw=6*s;
        nwyccw=2*s;
        nxccw=nxcfw;
        nyccw=nycfw;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxccw, nwyccw, nxccw,
              nyccw, q);
      }
        {/* south cut box */
        nwxccs=2*s;
        nwyccs=6*s;
        nxccs=nxcfs;
        nyccs=nycfs;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxccs, nwyccs, nxccs,
              nyccs, q);
      }
        {/* north cut box */
        nwxccn=2*s;
        nwyccn=6*s;
        nxccn=nxcfn;
        nyccn=nycfn;
```

```
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxccn, nwyccn, nxccn,
              nyccn, q);
        }
        {/* east cut box */
        nwxcce=6*s;
        nwycce=2*s;
        nxcce=nxcfe;
        nycce=nycfe;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcce, nwycce, nxcce,
              nycce, q);
        }
      fclose(ptr);

/* metal box */
      ptr = fopen("LCM_box", "a");
        {/* middle metal */
        nwxcmm=nwxcfm;
        nwycmm=nwycfm;
        nxcmm=nxcfm;
        nycmm=nycfm;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcmm, nwycmm, nxcmm,
              nycmm, q);
        }
        {/* west metal */
        nwxcmw=nwxcfw;
        nwycmw=nwycfw;
        nxcmw=nxcfw;
        nycmw=nycfw;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcmw, nwycmw, nxcmw,
              nycmw, q);
        }
        {/* south metal */
        nwxcms=nwxcfs;
        nwycms=nwycfs;
        nxcms=nxcfs;
        nycms=nycfs;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcms, nwycms, nxcms,
              nycms, q);
        }
        {/* north metal */
        nwxcmn=nwxcfn;
        nwycmn=nwycfn;
        nxcmn=nxcfn;
        nycmn=nycfn;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcmn, nwycmn, nxcmn,
              nycmn, q);
        }
        {/* east metal */
        nwxcme=nwxcfe;
        nwycme=nwycfe;
        nxcme=nxcfe;
        nycme=nycfe;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcme, nwycme, nxcme,
              nycme, q);
```

```
        }
        {/* connection metal to north contact */
         nwxcmcn=8*s+(nwxcfv-nwxcfn)/2;
         nwycmcn=2*s;
         nxcmcn=nwxcmcn/2+s+x;
         nycmcn=nycfn;
         fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcmcn, nwycmcn,
              nxcmcn, nycmcn, q);
        }
        {/* connection metal to west contact */
         nwxcmcw=2*s;
         nwycmcw=10*s+(nwycfh-nwycfw)/2;
         nxcmcw=x;
         nycmcw=(-1)*(nwycmcw/2+6*s)+y;
         fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcmcw, nwycmcw,
              nxcmcw, nycmcw, q);
        }
        {/* connection metal to GND */
         nwxcmcg=2*s;
         nwycmcg=16*s+(nwycfh-nwycfw)/2;
         nxcmcg=x;
         nycmcg=(-1)*(nwycmcg/2+2*s)+nycfw;
         fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcmcg, nwycmcg,
              nxcmcg, nycmcg, q);
        }
        {/* connection metal to east contact */
         nwxcmce=2*s;
         nwycmce=16*s+(nwycfh-nwycfw)/2;
         nxcmce=nxcfe;
         nycmce=(-1)*(nwycmce/2+2*s)+nycfe;
         fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcmce, nwycmce,
              nxcmce, nycmce, q);
        }
        {/* connection metal to middle contact */
         nwxcmcm=(nwxcfh-nwxcfm)/2-2*s;
         nwycmcm=2*s;
         nxcmcm=nxcfm+2*s+nwxcmcm/2;
         nycmcm=nycfe;
         fprintf(ptr,"%c %d %d %d %d%c0, b, nwxcmcm, nwycmcm,
              nxcmcm, nycmcm, q);
        }
        {/* p-n connection metal */
         nwxcm=2*s;
         nwycm=19*s+(nwycfh-nwycfm)/2;
         nxcm=nxbpe+s;
         nycm=(-1)*nwycm/2+y;
         fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcm, nwycm,
              nxcm, nycm, q);
        }
      fclose(ptr);

/* p-pluse */
      ptr = fopen("LCPP_box", "a");
        {/* west  box */
```

```
        nwxppw=6*s;
        nwyppw=8*s;
        nxppw=x-3*s;
        nyppw=nycfw;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxppw, nwyppw, nxppw,
             nyppw, q);
      }
      {/* south box */
        nwxpps=8*s;
        nwypps=6*s;
        nxpps=nxcfs;
        nypps=nycfs-nwypps/2;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxpps, nwypps, nxpps,
             nypps, q);
      }
      {/* north box */
        nwxppn=12*s;
        nwyppn=6*s;
        nxppn=nxcfn-2*s;
        nyppn=(-1)*(nwyppn/2+s);
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxppn, nwyppn, nxppn,
             nyppn, q);
      }
      {/* east box */
        nwxppe=6*s;
        nwyppe=8*s;
        nxppe=nxcfe+nwxppe/2;
        nyppe=nycfe;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxppe, nwyppe, nxppe,
             nyppe, q);
      }
    fclose(ptr);

/*n-pluse */
    ptr = fopen("LCNP_box", "a");
      {/* west  box */
        nwxnpw=nwxppw+2*s;
        nwynpw=nwyppw+4*s;
        nxnpw=nxppw-s;
        nynpw=nycfw;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxnpw, nwynpw, nxnpw,
             nynpw, q);
      }
      {/* south box */
        nwxnps=nwxpps+4*s;
        nwynps=nwypps+2*s;
        nxnps=nxpps;
        nynps=nypps-s;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxnps, nwynps, nxnps,
             nynps, q);
      }
      {/* north box */
        nwxnpn=nwxppn+4*s;
        nwynpn=nwynps;
```

```
    xcmo=wxcmo/2+nxcm-s;
    ycmo=y;
   fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmo, wycmo, xcmo, ycmo, q);
   }
/* VDD box */
   {
    if (wxnp<nwxng)
     wxvdd=nwxng;
    else
     wxvdd=10*s+wxnp;
     wyvdd=4*s;
     xvdd=nxng;
     yvdd=ycfn+2*s;
   fprintf(ptr, "%c %d %d %d %d%c0, b, wxvdd, wyvdd, xvdd, yvdd, q);
   }
/* GND box */
   {
    wxgnd=nwxng;
    wygnd=4*s;
    xgnd=nxng;
    ygnd=nynps-5*s;
   fprintf(ptr, "%c %d %d %d %d%c0, b, wxgnd, wygnd, xgnd, ygnd, q);
   }
   fclose(ptr);

/* output cut box */
/*   ptr=fopen("LCC_box", "a");
   {
    wxcco=2*s;
    wycco=2*s;
    xcco=wxcmo+x;
    ycco=y;
   fprintf(ptr, "%c %d %d %d %d%c0, b, wxcco, wycco, xcco, ycco, q);
   }
*/
/* input poly */
    ptr=fopen("Poly_box", "a");
    {
    wxbpe=15*s;
    wybpe=2*s;
    xbpe=x-2*s;
    ybpe=y+5*s;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxbpe, wybpe, xbpe, ybpe, q);
    }
   fclose(ptr);

/* output contact poly box */
/*   {
    wxbpo=6*s;
    wybpo=6*s;
    xbpo=xcco;
    ybpo=ycco;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxbpo, wybpo, xbpo, ybpo, q);
    }
```

```
   fclose(ptr);
*/
}


/* A module to create the parallel configuration of PMOS and NMOS inverters */
#include <stdio.h>
#include <math.h>
fun(s)
int s;
{
 extern int n1, m1, ycc1, nwycf, xcmo, nxcc, nycc;
 FILE *ptr, *fopen();
 int x=0, y=0;

 char b='B';
 char q=';';

 int  wxpp, wypp, xpp, ypp;
 int  wxnp, wynp, xnp, ynp;
 int  wxpw, wypw, xpw, ypw;
 int  wxpg, wypg, xpg, ypg;
 int  wxng, wyng, xng, yng;
 int  wxcm, wycm, xcm, ycm;
 int  wxcmo, wycmo, ycmo;
 int  nwxcm, nwycm, nxcm, nycm;
 int  wxcmi, wycmi, xcmi, ycmi;
 int  nwxpp, nwypp, nxpp, nypp;
 int  nwxnp, nwynp, nxnp, nynp;
 int  wxcc, wycc, xcc, ycc;
 int  wxcco, wycco, xcco, ycco;
 int  wxpb, wypb, xpb, ypb;
 int  wxpbo, wypbo, xpbo, ypbo;

/* p-pluse box */
   ptr=fopen("LCPP_box", "a");
    { /* p-transistor */
    if (n1%2==0)
    wxpp=xcmo+9*s;
    else
    wxpp=xcmo+6*s;
    wypp=ycc1-3*s;
    xpp=wxpp/2-4*s+x;
    ypp=wypp/2+3*s+y;
   fprintf(ptr, "%c %d %d %d %d%c0, b, wxpp, wypp, xpp, ypp, q);
    }

    {/* n-transistor */
    nwxpp=nxcc+8*s;
    nwypp=6*s;
    nxpp=nwxpp/2-4*s+x;
    nypp=nycc-nwypp/2;
   fprintf(ptr, "%c %d %d %d %d%c0, b, nwxpp, nwypp, nxpp, nypp, q);
    }
```

```
    fclose(ptr);

/* n-pluse box */
    ptr=fopen("LCNP_box", "a");
     {/* p-transistor */
     wxnp=wxpp+4*s;
     wynp=wypp+2*s;
     xnp=xpp;
     ynp=ypp-s;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxnp, wynp, xnp, ynp, q);
     }

     {/* n-transistor */
     nwxnp=nwxpp+4*s;
     nwynp=nwypp+2*s;
     nxnp=nxpp;
     nynp=nypp-s;
    fprintf(ptr, "%c %d %d %d %d%c0, b, nwxnp, nwynp, nxnp, nynp, q);
     }
    fclose(ptr);

/* p-well box */
    ptr=fopen("LCPW_box", "w");
     {
     wxpw=wxpp;
     wypw=abs(nycc)+3*s;
     xpw=xpp;
     ypw=(-1)*(wypw/2+3*s)+y;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxpw, wypw, xpw, ypw, q);
/*    printf("wxpw=%d wypw=%d xpw=%d ypw=%d0, wxpw, wypw, xpw, ypw); */
     }
    fclose(ptr);

/* p-gard box */
    ptr=fopen("LCPG_box", "w");
     {
     wxpg=wxpw+4*s;
     wypg=wypw+4*s;
     xpg=xpw;
     ypg=ypw;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxpg, wypg, xpg, ypg, q);
     }
    fclose(ptr);

/* n-gard box */
    ptr=fopen("LCNG_box", "w");
     {
     wxng=wxpg+4*s;
     wyng=wypg+4*s;
     xng=xpg;
     yng=ypg;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxng, wyng, xng, yng, q);
     }
    fclose(ptr);
```

```
/* VDD metal box */
    ptr=fopen("LCM_box", "a");
     {
     wxcm=wxng;
     wycm=6*s;
     xcm=xnp;
     ycm=wynp+wycm/2+s;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxcm, wycm, xcm, ycm, q);
     }

/* GND metal box */
     {
     nwxcm=wxcm;
     nwycm=wycm;
     nxcm=xng;
     nycm=(-1)*(wyng-7*s+nwycm/2);
    fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcm, nwycm, nxcm, nycm, q);
     }

/* input metel box */
     {
     wxcmi=4*s;
     wycmi=4*s;
     xcmi=(-1)*6*s+x;
     ycmi=y;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmi, wycmi, xcmi, ycmi, q);
     }
/* output metal box */
    {
    if (n1%2==0)
     wxcmo=6*s;
    else
     wxcmo=3*s;
    wycmo=4*s;
    xcmo=xcmo+2*s+wxcmo/2;
    ycmo=y;
   fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmo, wycmo, xcmo, ycmo, q);
    }
    fclose(ptr);

    ptr=fopen("LCC_box", "a");

/* input cut box */
     {
     wxcc=2*s;
     wycc=2*s;
     xcc=xcmi;
     ycc=ycmi;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxcc, wycc, xcc, ycc, q);
     }
/* output cut box */
     {
     wxcco=2*s;
     wycco=2*s;
```

```
        if (n1%2==0)
        xcco=xcmo+s;
        else
        xcco=xcmo-s/2;
        ycco=y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcco, wycco, xcco, ycco, q);
        }
      fclose(ptr);

      ptr=fopen("Poly_box", "a");

/* input poly box */
        {
        wxpb=6*s;
        wypb=6*s;
        xpb=xcc;
        ypb=y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxpb, wypb, xpb, ypb, q);
        }
/* output poly box */
        {
        wxpbo=6*s;
        wypbo=6*s;
        xpbo=xcco;
        ypbo=y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxpbo, wypbo, xpbo, ypbo, q);
        }
      fclose(ptr);
}


/* A module to connect both PMOS and NMOS transistors */
#include <strings.h>
#include <stdio.h>
#include <math.h>
fun0(s, axis2)
int s;
int axis2;
{
  extern int wycf, nxcf, nwxcf, nwycf, nycf;
  FILE *ptr, *fopen();
  int x=0, y=0;

  char b='B';
  char q=';';

  int  wxpp, wypp, xpp, ypp;
  int  wxnp, wynp, xnp, ynp;
  int  wxpw, wypw, xpw, ypw;
  int  wxpg, wypg, xpg, ypg;
  int  wxng, wyng, xng, yng;
  int  wxcm, wycm, xcm, ycm;
  int  nwxcm, nwycm, nxcm, nycm;
  int  wxcmi, wycmi, xcmi, ycmi;
```

```
 int  wxcc, wycc, xcc, ycc;
 int  wxpb, wypb, xpb, ypb;

/* p-pluse box */
    ptr=fopen("LCPP_box", "a");
      {
      wxpp=abs(axis2)-4*s;
      wypp=abs(wycf)-2*s;
      xpp=wxpp/2-4*s+x;
      ypp=wypp/2+s+y;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxpp, wypp, xpp, ypp, q);
/*     printf("wxpp=%d wypp=%d xpp=%d ypp=%d0, wxpp, wypp, xpp, ypp); */
      }
    fclose(ptr);

/* n-pluse box */
    ptr=fopen("LCNP_box", "a");
      {
      wxnp=wxpp+4*s;
      wynp=wypp+2*s;
      xnp=xpp;
      ynp=ypp-s;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxnp, wynp, xnp, ynp, q);
      }
    fclose(ptr);

/* p-well box */
    ptr=fopen("LCPW_box", "w");
      {
      wxpw=abs(nxcf)+abs(nwxcf)/2+7*s;
      wypw=abs(nwycf)+5*s;
      xpw=wxpw/2-4*s+x;
      ypw=(-1)*(wypw/2+4*s)+y;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxpw, wypw, xpw, ypw, q);
/*     printf("wxpw=%d wypw=%d xpw=%d ypw=%d0, wxpw, wypw, xpw, ypw); */
      }
    fclose(ptr);

/* p-gard box */
    ptr=fopen("LCPG_box", "w");
      {
      wxpg=wxpw+4*s;
      wypg=wypw+4*s;
      xpg=xpw;
      ypg=ypw;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxpg, wypg, xpg, ypg, q);
      }
    fclose(ptr);

/* n-gard box */
    ptr=fopen("LCNG_box", "w");
      {
      wxng=wxpg+4*s;
      wyng=wypg+4*s;
```

```
        xng=xpg;
        yng=ypg;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxng, wyng, xng, yng, q);
        }
      fclose(ptr);

/* VDD metal box */
      ptr=fopen("LCM_box", "a");
        {
        wxcm=wxnp+6*s;
        wycm=6*s;
        xcm=xnp-s;
        ycm=wynp-s+wycm/2;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcm, wycm, xcm, ycm, q);
/*      printf("wxcm=%d wycm=%d xcm=%d ycm=%d0, wxcm, wycm, xcm, ycm); */
        }

/* GND metal box */
        {
        nwxcm=wxcm;
        nwycm=wycm;
        nxcm=nwxcm/2-10*s;
        nycm=(-1)*(wyng-6*s+nwycm/2);
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcm, nwycm, nxcm, nycm, q);
/*    printf("nwxcm=%d nwycm=%d nxcm=%d nycm=%d0, nwxcm, nwycm, nxcm, nycm); */
        }

/* input metel box */
        {
        wxcmi=4*s;
        wycmi=4*s;
        xcmi=(-1)*7*s+x;
        ycmi=y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcmi, wycmi, xcmi, ycmi, q);
        }
      fclose(ptr);

      ptr=fopen("LCC_box", "a");
/* input cut box */
        {
        wxcc=2*s;
        wycc=2*s;
        xcc=xcmi;
        ycc=ycmi;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcc, wycc, xcc, ycc, q);
        }
      fclose(ptr);

      ptr=fopen("Poly_box", "a");
/* input poly box */
        {
        wxpb=6*s;
        wypb=6*s;
        xpb=xcc;
```

```
    ypb=y;
    fprintf(ptr, "%c %d %d %d %d%c0, b, wxpb, wypb, xpb, ypb, q);

    fclose(ptr);
}


/* A module to create a PMOS transistor in zig zag structure */
#include <stdio.h>
#include <math.h>
int wycf;
pmos(wp, threshold, axis2, beta, s)
float beta;
int wp, threshold, axis2, s;
{
 int x=0, y=0;

  FILE *ptr, *fopen();
  char b='B';
  char q=';';
  int wxbp, wxsp, wxcf, wxpp, wxnp, wxcc, wxcm;
  int wybp, wysp, wypp, wynp, wycc, wycm;
  int xbp, xsp, xcf, xpp, xnp, xcc, xcm;
  int ybp, ysp, ycf, ypp, ynp, ycc, ycm;
  int wxcm1, wycm1, xcm1, ycm1;
  int wxcc1, wycc1,xcc1, ycc1;
  int wxpe, wype, xpe, ype;
  int rest;


  x=s+abs(axis2);
  rest=wp-threshold;
  if (threshold%2==1)
    {
     threshold=threshold+1;
    }

   ptr = fopen("Poly_box", "a");
    {
       wxbp=2*s;
       wybp=threshold;
       xbp=x;
       ybp=8*s+wybp/2+y;
       fprintf(ptr, "%c %d %d %d %d%c0, b, wxbp, wybp, xbp, ybp, q);
      }
/* horizatal poly box */

      {
      if (rest > 5*s)
       wxsp=rest;
      else
       wxsp=5*s;
       wysp=2*s;
       xsp=wxsp/2+wxbp/2+x;
```

```
        ysp=wybp+7*s+y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxsp, wysp, xsp; ysp, q);
      }
        { /* last out of CF poly box to extension */
        wxpe=2*s;
        wype=2*s;
        xpe=wxsp+wxpe/2+wxbp/2+x;
        ype=ysp;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxpe, wype, xpe, ype, q);
        }
      fclose(ptr);

/* device well box */
      ptr = fopen("LCF_box", "a");
        {
        wxcf=wxbp+wxsp+2*s;
        wycf=wybp+14*s;
        xcf=wxcf/2-3*s+x;
        ycf=3*s+wycf/2+y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxcf, wycf, xcf, ycf, q);
        }
      fclose(ptr);

/* p-pulse box */
/*      ptr = fopen("LCPP_box", "a");
        {
        wxpp=wxcf+6*s;
        wypp=wycf-2*s;
        xpp=xcf+s;
        ypp=s+wypp/2+y;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxpp, wypp, xpp, ypp, q);
        }
      fclose(ptr);
*/
/* n-pulse box */
/*      ptr = fopen("LCNP_box", "a");
        {
        wxnp=wxpp+4*s;
        wynp=wypp+2*s;
        xnp=xpp;
        ynp=ypp-s;
      fprintf(ptr, "%c %d %d %d %d%c0, b, wxnp, wynp, xnp, ynp, q);
        }
      fclose(ptr);
*/
/* contect cut box */
      ptr = fopen("LCC_box", "a");
      /* upper cut box */
        {
        wxcc1=2*s;
        wycc1=6*s;
        xcc1=wxcf/2-3*s+x;
        ycc1=wycf-s+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, wxcc1, wycc1, xcc1, ycc1, q);
```

```
        }
      /* below cut box */
        {
          wxcc=2*s;
          wycc=2*s;
          xcc=4*s+x;
          ycc=5*s+y;
          fprintf(ptr, "%c %d %d %d %d%c0, b, wxcc, wycc, xcc, ycc, q);
        }
      fclose(ptr);

/* metal box */
      ptr = fopen("LCM_box", "a");
      /* upper metal */
        {
          wxcm1=2*s+wxcc1;
          wycm1=2*s+wycc1;
          xcm1=xcc1;
          ycm1=ycc1;
          fprintf(ptr, "%c %d %d %d %d%c0, b, wxcm1, wycm1, xcm1, ycm1, q);
        }
      /* below metal */
        {
          wxcm=wxcc+2*s;
          wycm=wycc+2*s;
          xcm=xcc;
          ycm=ycc;
          fprintf(ptr, "%c %d %d %d %d%c0, b, wxcm, wycm, xcm, ycm, q);
        }
      fclose(ptr);

    return(wxcf);
}


/* A module to create an NMOS transistor in zig zag structure */
#include <stdio.h>
#include <math.h>
int nxcf, nwxcf, nwycf, nycf;
nmos(wn, threshold, axis2, beta, s)
float beta;
int wn, threshold, axis2, s;
{
  int x=0, y=0;

  int c, i, j;
  FILE *ptr, *fopen();
  char Poly_box;
  char b='B';
  char q=';';

  int nwxbp, nwxsp, nwxpp, nwxnp, nwxcc, nwxcm, nwxpg, nwxng;
  int nwybp, nwysp, nwypp, nwynp, nwycc, nwycm, nwypg, nwyng;
  int nxbp, nxsp, nxpp, nxnp, nxcc, nxcm, nxpg, nxng;
```

```
int nybp, nysp, nypp, nynp, nycc, nycm, nypg, nyng;
int nwxcc1, nwycc1, nxcc1, nycc1, nwxcm1, nwycm1, nxcm1, nycm1;

int wxpe, wype, xpe, ype, nwxpe, nwype, nxpe, nype;
int wxbpe, wybpe, xbpe, ybpe;
int rest;

x==s+abs(axis2);
rest==wn-threshold;
if (threshold%2===1)
  {
   threshold==threshold+1;
  }

/* ntransistor */
   ptr == fopen("Poly_box", "a");
   /* big poly box */
    {
     nwxbp==2*s;
     nwybp==threshold;
     nxbp==x;
     nybp==(-1)*(nwybp/2+12*s)+y;
     fprintf(ptr, "%c %d %d %d %d%c0, b, nwxbp, nwybp, nxbp, nybp, q);
    }
   /* horizatal poly box */
    {
    if (rest > 5*s)
     nwxsp==rest;
    else
     nwxsp==5*s;
     nwysp==2*s;
     nxsp==nwxsp/2+nwxbp/2+x;
     nysp==(-1)*(nwybp+11*s)+y;
     fprintf(ptr, "%c %d %d %d %d%c0, b, nwxsp, nwysp, nxsp, nysp, q);
    }
     {/* ntrsistor big poly extension */
     nwxpe==2*s;
     nwype==2*s;
     nxpe==nwxsp+nwxbp/2+nwxpe/2+x;
     nype==nysp;
     fprintf(ptr, "%c %d %d %d %d%c0, b, nwxpe, nwype, nxpe, nype, q);
     }
   fclose(ptr);

/* device-well box */
   ptr == fopen("LCF_box", "a");
    {
    nwxcf==nwxbp+nwxsp+2*s;
    nwycf==nwybp+14*s;
    nxcf==nwxcf/2-3*s+x;
    nycf==(-1)*(nwycf/2+7*s)+y;
  fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcf, nwycf, nxcf, nycf, q);
    }
  fclose(ptr);
```

```
/* p-well box */
/*      ptr = fopen("LCPW_box", "a");
       {
       nwxpw=nwxcf+4*s;
       nwypw=nwycf+4*s;
       nxpw=nxcf;
       nypw=nycf;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxpw, nwypw, nxpw, nypw, q);
       }
      fclose(ptr);
*/
/* p-gard box */
/*      ptr = fopen("LCPG_box", "a");
       {
       nwxpg=nwxpw+4*s;
       nwypg=nwypw+4*s;
       nxpg=nxcf;
       nypg=nycf;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxpg, nwypg, nxpg, nypg, q);
       }
      fclose(ptr);
*/
/* n-gard box */
/*      ptr = fopen("LCNG_box", "a");
       {
       nwxng=nwxpg+4*s;
       nwyng=nwypg+4*s;
       nxng=nxcf;
       nyng=nycf;
      fprintf(ptr, "%c %d %d %d %d%c0, b, nwxng, nwyng, nxng, nyng, q);
       }
      fclose(ptr);
*/
/* contect cut box */
      ptr = fopen("LCC_box", "a");
      /* upper contect */
       {
        nwxcc1=2*s;
        nwycc1=2*s;
        nxcc1=4*s+x;
        nycc1=(-1)*9*s+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcc1, nwycc1, nxcc1,
              nycc1, q);
       }
      /* below contect */
       {
        nwxcc=2*s;
        nwycc=6*s;
        nxcc=2*s+x;
        nycc=(-1)*(nwybp+14*s+nwycc/2)+y;
        fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcc, nwycc, nxcc, nycc, q);
       }
      fclose(ptr);
```

```
/* metal box */
     ptr = fopen("LCM_box", "a");
     /* upper contect */
      {
       nwxcm1=nwxcc1+2*s;
       nwycm1=nwycc1+2*s;
       nxcm1=nxcc1;
       nycm1=nycc1;
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcm1, nwycm1, nxcm1,
            nycm1, q);
      }
     /* below contect */
      {
       nwxcm=2*s+nwxcc;
       nwycm=nwycc+2*s;
       nxcm=nxcc;
       nycm=nycc;
       fprintf(ptr, "%c %d %d %d %d%c0, b, nwxcm, nwycm, nxcm, nycm, q);
      }
     fclose(ptr);

/* p-pulse box */
     ptr = fopen("LCPP_box", "a");
      {
      nwxpp=nwxcf+4*s;
      nwypp=6*s;
      nxpp=nxcf;
      nypp=nycc-nwypp/2;
     fprintf(ptr, "%c %d %d %d %d%c0, b, nwxpp, nwypp, nxpp, nypp, q);
      }
     fclose(ptr);

/* n-pulse box */
     ptr = fopen("LCNP_box", "a");
      {
      nwxnp=nwxpp+4*s;
      nwynp=nwypp+2*s;
      nxnp=nxpp;
      nynp=nycc-nwynp/2;
     fprintf(ptr, "%c %d %d %d %d%c0, b, nwxnp, nwynp, nxnp, nynp, q);
      }
     fclose(ptr);
}


/* A module to generate CIF file */
#include <stdio.h>
char string0[]="(CIF file of symbol hierarchy rooted at";
char string1[]="DS 1 1 1;";
char string2[]="9";
char string3[]="L CP;";
char string4[]="L CF;";
char string5[]="L CPP;";
char string6[]="L CNP;";
```

```
char string7[]="L CC;";
char string8[]="L CM;";
char string9[]="L CPW;";
char string10[]="L CPG;";
char string11[]="L CNG;";
char string12[]="DF;";
char string13[]="C 1;";
char string14[]="E";
char string15[]=");";
cifgen(name)
char name[];
 {
   FILE *ptr, *fp, *fopen();
   int c;
   char q=';';

   ptr = fopen(name, "w");
       {
        fprintf(ptr, "%s %s %s\n", string0, name, string15);
        fprintf(ptr, "%s\n", string1);
        fprintf(ptr, "%s %s %c\n", string2, name, q);
        fprintf(ptr, "%s\n", string3);

        fp = fopen("Poly_box", "r");
          {
           while ((c=getc(fp))!=EOF)
           putc (c, ptr);
            fclose(fp);
          }

        fprintf(ptr, "%s\n", string4);

        fp = fopen("LCF_box", "r");
          {
           while ((c=getc(fp))!=EOF)
           putc (c,ptr);
            fclose(fp);
          }

        fprintf(ptr, "%s\n", string5);

        fp = fopen("LCPP_box", "r");
          {
          while ((c=getc(fp))!=EOF)
          putc (c,ptr);
           fclose(fp);
          }

        fprintf(ptr, "%s\n", string6);

        fp = fopen("LCNP_box", "r");
          {
          while ((c=getc(fp))!=EOF)
          putc (c,ptr);
```

```
    fclose(fp);
   }

fprintf(ptr, "%s\n", string7);

fp = fopen("LCC_box", "r");
  {
  while ((c=getc(fp))!=EOF)
  putc (c,ptr);
    fclose(fp);
  }

fprintf(ptr, "%s\n", string8);

fp = fopen("LCM_box", "r");
  {
  while ((c=getc(fp))!=EOF)
  putc (c,ptr);
    fclose(fp);
  }

fprintf(ptr, "%s\n", string9);

fp = fopen("LCPW_box", "r");
  {
  while ((c=getc(fp))!=EOF)
  putc (c,ptr);
    fclose(fp);
  }

fprintf(ptr, "%s\n", string10);

fp = fopen("LCPG_box", "r");
  {
  while ((c=getc(fp))!=EOF)
  putc (c,ptr);
    fclose(fp);
  }

fprintf(ptr, "%s\n", string11);

fp = fopen("LCNG_box", "r");
  {
  while ((c=getc(fp))!=EOF)
  putc (c,ptr);
    fclose(fp);
  }

fprintf(ptr, "%s\n", string12);
fprintf(ptr, "%s\n", string13);
fprintf(ptr, "%s\n", string14);

fclose(ptr);
}
```

```
    return;
  }


/* A module to generate KIC file */
#include <stdio.h>
char strin0[]="(Symbol";
char strin1[]="DS 0 1 1;";
char strin2[]="9";
char strin3[]="L CP;";
char strin4[]="L CF;";
char strin5[]="L CPP;";
char strin6[]="L CNP;";
char strin7[]="L CC;";
char strin8[]="L CM;";
char strin9[]="L CPW;";
char strin10[]="L CPG;";
char strin11[]="L CNG;";
char strin12[]="DF;";
char strin14[]="E";
char strin15[]=");";
kicgen(name)
char name[];
 {
    FILE *ptr, *fp, *fopen();
    int c;
    char q=';';

    ptr = fopen(name, "w");
        {
         fprintf(ptr, "%s %s %s\n", strin0, name, strin15);
         fprintf(ptr, "%s %s %c\n", strin2, name, q);
         fprintf(ptr, "%s\n", strin1);
         fprintf(ptr, "%s\n", strin3);

         fp = fopen("Poly_box", "r");
          {
           while ((c=getc(fp))!=EOF)
           putc (c, ptr);
             fclose(fp);
          }

         fprintf(ptr, "%s\n", strin4);

         fp = fopen("LCF_box", "r");
          {
           while ((c=getc(fp))!=EOF)
           putc (c,ptr);
             fclose(fp);
          }

         fprintf(ptr, "%s\n", strin5);

         fp = fopen("LCPP_box", "r");
```

```
 {
 while ((c=getc(fp))!=EOF)
 putc (c,ptr);
  fclose(fp);
 }

fprintf(ptr, "%s\n", strin6);

fp = fopen("LCNP_box", "r");
 {
 while ((c=getc(fp))!=EOF)
 putc (c,ptr);
  fclose(fp);
 }

fprintf(ptr, "%s\n", strin7);

fp = fopen("LCC_box", "r");
 {
 while ((c=getc(fp))!=EOF)
 putc (c,ptr);
  fclose(fp);
 }

fprintf(ptr, "%s\n", strin8);

fp = fopen("LCM_box", "r");
 {
 while ((c=getc(fp))!=EOF)
 putc (c,ptr);
  fclose(fp);
 }

fprintf(ptr, "%s\n", strin9);

fp = fopen("LCPW_box", "r");
 {
 while ((c=getc(fp))!=EOF)
 putc (c,ptr);
  fclose(fp);
 }

fprintf(ptr, "%s\n", strin10);

fp = fopen("LCPG_box", "r");
 {
 while ((c=getc(fp))!=EOF)
 putc (c,ptr);
  fclose(fp);
 }

fprintf(ptr, "%s\n", strin11);

fp = fopen("LCNG_box", "r");
```

```
        {
        while ((c=getc(fp))!=EOF)
        putc (c,ptr);
          fclose(fp);
        }

      fprintf(ptr, "%s\n", strin12);
      fprintf(ptr, "%s\n", strin14);

    fclose(ptr);
    }
}
```